12-2016

Three Body Interactions of Rare Gas Solids Calculated Within the Einstein Model

Dan D'Andrea
University of Tennessee, Knoxville, ddandrea@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Dan D'Andrea entitled "Three Body Interactions of Rare Gas Solids Calculated Within the Einstein Model." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Chemistry.

Robert J. Hinde, Major Professor

We have read this thesis and recommend its acceptance:

Charles Collins, Tessa Calhoun, Sharani Roy, Michael D. Best

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Three Body Interactions of Rare Gas Solids Calculated Within the Einstein Model

A Thesis Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Dan D'Andrea
December 2016

Abstract

Three body interactions can become important in solids at higher pressures and densities as the molecules can come into close contact. At low temperatures, accurate studies of three body interactions in solids require averaging the three-body terms over the molecules' zero point motions. An efficient, but approximate, averaging approach is based on a polynomial approximation of the three-body term. The polynomial approximation can be developed as a function of the symmetry coordinates of a triangle displaced from its average geometry and also as a function of the Cartesian zero point displacements from each atom's average position. The polynomial approximation approach can be checked through two more accurate, but more time-consuming methods: Gaussian quadrature or Monte Carlo integration of the exact three-body function. Results are presented for solid helium, solid neon and solid argon, treated as Einstein solids. An evaluation of the quality of the Einstein model approximation will also be presented. Results for helium will be compared with quantum Monte Carlo simulations.

Table of Contents

List of Tables

List of Figures

Acronyms

AGY- Audette Giese York
EAT- Extended Axilrod-Teller
FCC- Face Center Cubic
GQ- Gaussian Quadrature
GSM- Gough Smith Maitland
HCP- Hexagonal Closed Packed
KE- Kinetic Energy
LJ- Lennard Jones
MAT- Modified Axilrod-Teller
MCI- Monte Carlo Integration
MTT- Modified Tang Toennies
SET- Small Equilateral Triangle
SLT- Small Linear Triangle
TBE- Three Body Potential of Ermakova
TPA- The Polynomial Approximation
TT- Tang Toennies

Chapter 1: Introduction

Crystals are an interesting topic for theoretical chemists. The ability to use quantum chemical methods to compute the energy of a crystal would allow theoretical chemists to predict crystal structures, equations of states and phase diagrams. Therefore, finding ways to calculate crystal energies using quantum chemistry and test these calculations is important. It is common to test calculations of crystal energies by computing the cohesive energy, the difference of the per-molecule energies of the solid phase and the gas phase. The cohesive energy can be measured experimentally, however this measurement will include the effects of zero-point motion. On the other hand, when cohesive energy is calculated theoretically it usually does not include the zero-point motion. Therefore, we need estimates of zero-point energy to accurately compare computed and measured cohesive energies. The zero-point energy cannot be measured directly. In order to estimate the zero-point energy, a combination of spectroscopic and theoretical methods is typically used [1]. The zero-point energy can be determined theoretically; however, it is very important to evaluate the accuracy of these calculations.

Many factors can influence the zero-point energy. At high density, the three body interactions start to influence the zero-point energy. Three body interactions play a larger and larger role as you go higher in density. Thus, calculation of three body energies at higher densities will be most important. The three body energy can be important as it can be used to improve the accuracy of equations of states which relate pressure and density [23].
The major goal of this research is developing a way to accurately and quickly estimate the three body energy contribution to the zero-point energy. Three body energies of solid helium, neon and argon were calculated through the Einstein model with the utilization of three methods: Monte Carlo integration, Gaussian quadrature and an approach based on a polynomial approximation of the three body energy. The polynomial approximation approach has the potential to be very efficient, but we must test its accuracy.

Einstein Model

The Einstein model for the properties of an atomic crystal states that all the atoms in the crystal will oscillate at the same frequency, but do so independently [12]. Each atom is considered to behave like a three-dimensional harmonic oscillator with its rest position at the atom's lattice site. To illustrate this, we can think of a rubber band attached to the atom that pulls the atom back to its equilibrium lattice site as it vibrates or moves away from that position. Because the atoms oscillate independently, when one atom moves a neighboring atom does not necessarily have to move in a correlated fashion. In the Einstein model, the atomic motions are therefore completely uncorrelated. This simplification is one of the big benefits of the Einstein model, because an atom’s zero-point motion can be described by a three-dimensional Gaussian probability distribution. A disadvantage to the Einstein model is that it does not give a good dynamical description at $T > 0$ K, as can be shown by heat capacity data in reference 13.

However, the Einstein model makes a good starting point for the present study because it is shown to be a good structural description close to $T = 0$ K [14]. Therefore, to start, the Einstein model will be applied, but eventually more realistic models will be employed. Eventually

molecular crystals will be of interest, but before these future challenges can be explored, rare gas solids will serve as the bridge to more complicated systems.

The Debye-Waller factor $\langle U^2 \rangle$ is the mean squared displacement of an atom from its average position in the solid. This quantifies the range of motion that an atom undergoes around its base position or lattice site [10]. Experimentally $\langle U^2 \rangle$ can be measured using neutron or X-ray scattering [11]. When temperature is increased $\langle U^2 \rangle$ will also increase, because of the atoms' thermal motion. However, these thermal motions are unimportant in the present study, as the focus of our research is the behavior of rare gas solids at T = 0 K. Approximating the distribution of atomic displacement vectors $u=(u_x,u_y,u_z)$ by a normalized Gaussian $N\exp(-c(Ux^2+Uy^2+Uz^2))$ will yield the relationship below.

$$\langle U^2 \rangle = 3/2c \quad (1)$$

where c describes the size of the atom's zero-point motion, and therefore controls the Debye-Waller factor. When c is small the zero-point motion is large and vice versa. We can also write the distribution of displacement vectors as $N\exp(-(Ux^2+Uy^2+Uz^2)/2\alpha^2)$, and where the above equation can be manipulated to relate c and alpha.

$$2c = 1/\alpha^2 \quad (2)$$

The parameter alpha displays the opposite trend in comparison to c. As alpha gets larger then $\langle U^2 \rangle$ gets larger. A graphical representation of $\langle U^2 \rangle$ can be shown in Figures 1.1 and 1.2.

The Figures show how $\langle U^2 \rangle$ controls localization. When $\langle U^2 \rangle$is large the atoms are less localized around their average lattice site (see figure 1.1). A smaller $\langle U^2 \rangle$ shows much more localization around the average lattice site. Due to the small mass of the helium atom it has a fairly large Debye-Waller factor. However, the value of $\langle U^2 \rangle$ will change with the density of the helium crystal. As density increases the $\langle U^2 \rangle$ value becomes smaller, because the probability of exchange is low and as the helium atoms become more closely packed, the helium atoms restrict the movements of their neighbors. The $\langle U^2 \rangle$ values for helium used here were obtained from quantum Monte Carlo simulations done by Ashleigh Barnes [18] and the $\langle U^2 \rangle$ values for neon and argon were computed from Lindemann parameters given by Henry Glyde [8]. From $\langle U^2 \rangle$ we can determine the value for alpha and c in the distribution of displacement vectors using equations (1) and (2).

Rare Gas Solids

Helium

Solid helium can arrange itself in a hexagonal close packed (HCP), body centered cubic (BCC), or face centered cubic (FCC) crystal lattice depending on the pressure and temperature of the system. The phase diagram of helium can be observed in Figure 2 of reference [8].

At T = 0 K and atmospheric pressure ^{4}He would be a liquid, however by increasing the pressure to about 25 bar, a solid is obtained. This solid forms predominantly the HCP crystal lattice with a pocket of BCC structures visible in a small region of the phase diagram. In order to obtain the FCC lattice, the pressure would need to reach as high as 1000 bar. The superfluid region, which is basically liquid helium acting without viscosity, is even more interesting. This region can be found at pressures under 25 bar and temperature less than 2.17 K. Helium can also

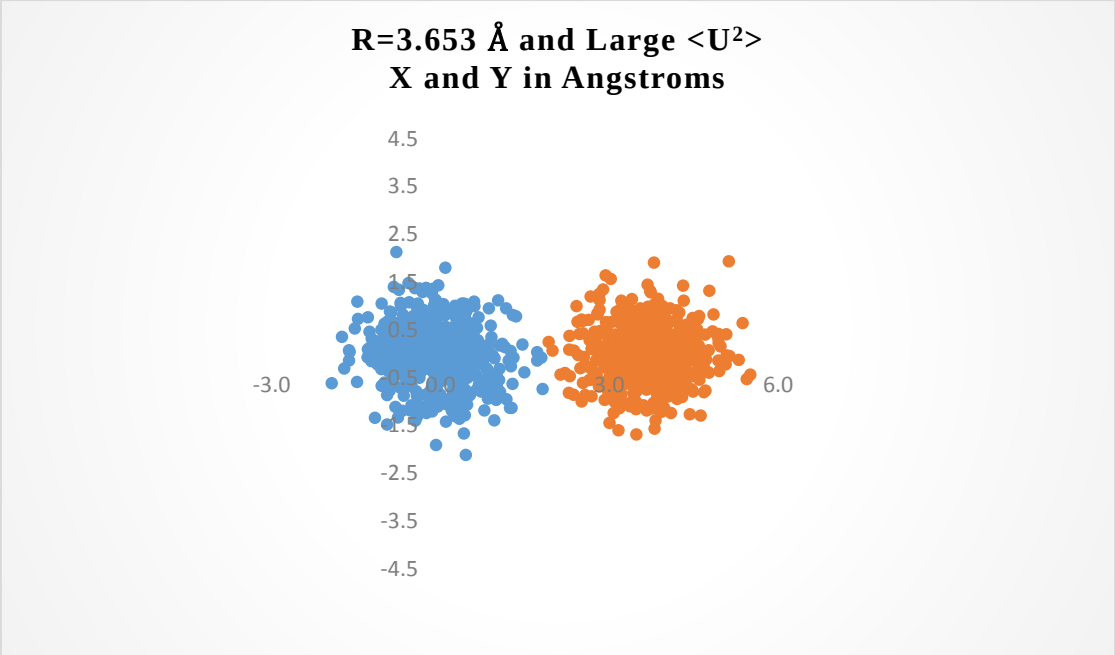

Figure 1.1: Large $<U^2>$

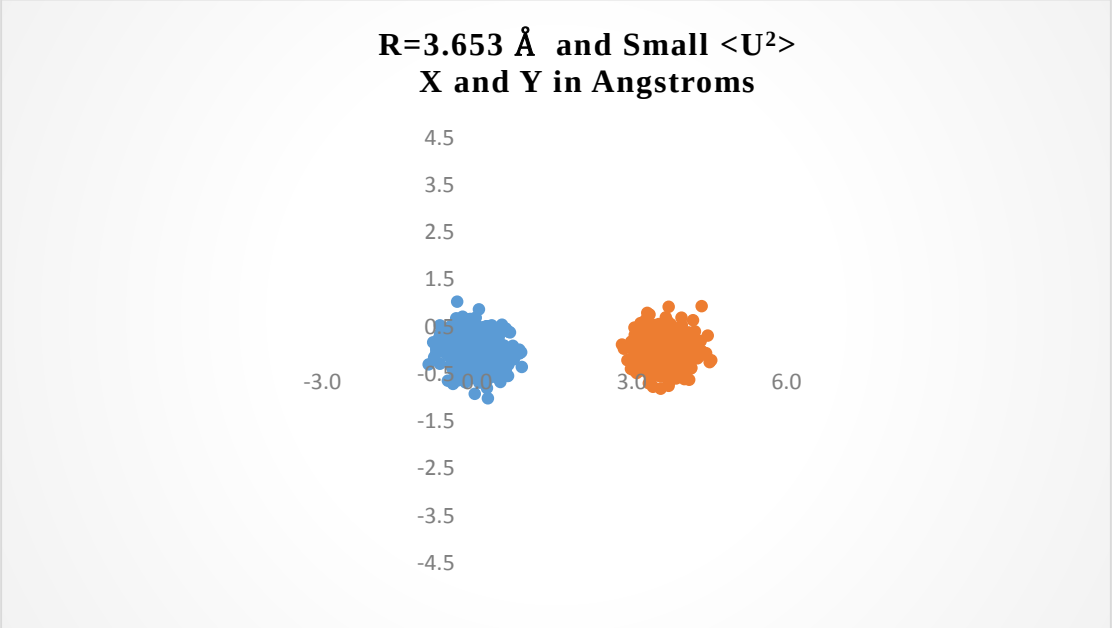

Figure 1.2: Small $<U^2>$

form a super solid (mixture of superfluid and solid properties) [9,10]. A study done by Kim and Chan [11,12] showed a superfluid fraction in solid helium present in pressures of up to 65 bar.

Neon and Argon

Solid neon arranges itself in the FCC crystal lattice. The melting temperature for neon is 24.55 K. Neon's phase diagram is less complicated than helium. At T=0 kelvin, neon is a solid without having to apply external pressure.

Solid argon also arranges itself in the FCC crystal lattice. Argon's melting temperature is 83.75 K. Argon is very similar to neon with respect to the phase diagram. Like neon, argon forms a solid at T=0 kelvin without having to apply external pressure.

Goals

A three body energy needs to be calculated through a fast method without sacrificing too much accuracy. If there was no zero-point motion the problem would be simplified to gathering the Cartesian coordinates of three given atoms or molecules in the crystal and then importing the coordinates to a three body energy function and then calculating the three body energy. However, the zero-point motion allows the atoms or molecules to move, so there is not a set position which they will occupy. In atomic solids, especially in helium, at absolute zero, zero-point energy tends to be substantial, meaning the vibrational amplitude of the atoms around their lattice sites is fairly large [2]. This creates a wider range of locations for where the helium atom could be found. A heavier molecule presents a bit of a different story. The zero-point motion of a heavy molecule is usually fairly small at absolute zero; however, the zero-point energy still plays a role, but does not dominate the way it does in an atomic solid. The density also plays an important role. At a low density, the solid will have a larger nearest neighbor distance giving the atoms or molecules more space. This allows the zero-point motion to be greater. As we go higher in density the nearest neighbor distance gets small and the atoms or molecules start to become more closely packed restricting the range of motion. Therefore, the zero-point motion becomes smaller as the density gets higher [9, 19, 21]. When we consider the size of the distribution or range of zero-point motion, the problem that we are interested in can really be thought of as the calculation of an average three body energy. Thus, we can think of our problem as a triple integral which can be observed below.

$$Avg\ E=\iiint E(r_1,r_2,r_3)P(r_1,r_2,r_3)dr_1dr_2dr_3 \quad (3)$$

where $E(r_1,r_2,r_3)$ is the three body energy as a function of r_1, r_2 and r_3 which are the Cartesian coordinates of the three different atoms or molecules. $P(r_1,r_2,r_3)$ are the probabilities of where you will find the atoms or molecules, where $r_1=(x_1, y_1, z_1)$.

This integral can be estimated through methods such as Gaussian quadrature and Monte Carlo integration. These are accurate methods, but they are also more time consuming. The real downside of these two methods is that a function $E(r_1,r_2,r_3)$ is needed in order to utilize either Gaussian quadrature or Monte Carlo integration. This works out nicely if rare gas solids were the only systems that were going to be studied, as there are available several three body potential energy functions designed for helium, neon and argon. Thus, when studying the rare gases both Gaussian quadrature and Monte Carlo integration are very convenient approaches as a three body energy function can be applied easily to each of these methods. However, what about a solid composed of molecules such as H_2? It is still a fairly simple system, however there actually is no three body potential energy function known yet for H_2.

This is the real challenge, how do we evaluate the average three body energy given by Eq. (3), if there currently is no three body energy potential function for the system? One possible option is to build a three body energy function, possibly from quantum chemical calculations. However, while that would help solve the problem of obtaining a function, building one would be very tedious and time consuming to the extent that it would really slow down the research. The better strategy would be one that allows us to bypass the function all together. This provides the possibility of analyzing more systems' three body energies at a much faster pace.

In order to achieve this a faster method will be introduced using a Theoretical Polynomial Approximation (TPA) for $E(r_1,r_2,r_3)$. The three body energy function can then instead be written as a polynomial. This is why this method does not need a function to calculate the three body energy, because the TPA is a function in its own right. Here we use an Einstein model for the zero-point motion of rare gas solids to explore this approach. The flexibility and simplicity of the Einstein model allows us to simulate rare gas solids with either small or large zero point motions just by varying the parameters. Therefore, the major goal of this research is to develop and test methods for quickly and accurately evaluating the three-body contribution to the zero-point energy of atomic and molecular crystals. Methods based on a TPA of the three-body energy have the potential to achieve this goal and allow the investigation of a wide range of systems.

Chapter 2: Literature Review

Phase Diagrams of Rare Gases

While neon and argon primarily form crystals with FCC configurations, helium is more interesting. It typically forms crystals with the HCP configuration, however depending on pressure and temperature solid helium can also exhibit FCC or BCC configurations. Three body energy may play a role in the stability of these configurations. In the future H_2 could be of interest; its phase diagram can be observed in figure 23 of reference 34. The figure shows how there are two different configurations at different pressures [34]. A three body energy may help explain why one configuration is favored than another configuration.

Cohesive Energy of Rare Gases

As mentioned above a cohesive energy can be measured experimentally, but this measurement includes the effects of zero-point motion. Experimental values for the cohesive energies of both argon and neon were obtained in the limit of zero pressure and T = 0 K.

Under these conditions the nearest neighbor distance for solid neon is 5.92 bohr [30], and that for solid argon is 7.10 bohr [35]. The experimental cohesive energies can be observed below in table 2.1. We will compare our results with these experimental values.

Table 2.1: Experimental Cohesive Energy

Atom	Nearest Neighbor Distance (bohr)	Cohesive Energy (Hartrees)
Neon	5.92	-7.163E-04
Argon	7.10	-2.938E-03

Potential Energy functions

Two Body Potentials

The two body energy potentials utilized were the Korona/TT (for neon only) [27], Lennard Jones (for argon and neon) [24], [n(r)]-6 potential also known as the GSM (for argon only) [25], AGY potential (for argon and neon) [29], Aziz-Slaman potential (for argon only) [26] and a modified TT (for neon only) [28].

The Lennard Jones potential is very simplistic which makes it ideal for computer simulations. This function yields the intermolecular potential between two atoms. The form of the potential can be shown in equation 4.

$$V(r) = 4\varepsilon[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}] \quad (4)$$

where V(r) is the intermolecular potential, r is the distance between two atoms, ε is the well depth, and σ is the distance at which the intermolecular potential between the two atoms reaches zero [24]. The parameters for ε and σ can be shown below in table 2.2.

Table 2.2: Parameters for Lennard Jones

Lennard Jones Potential Parameters		
Parameter	**Sigma (A.U.)**	**Epsilon (A.U.)**
Argon	6.323	3.981 E-04
Neon	5.259	1.180 E-04

The GSM potential is very similar to the Lennard-Jones potential. Its form can be observed below in equation 5.

$$V(r) = \epsilon[\frac{6}{n-6}r^{-n} - \frac{n}{n-6}r^{-6}] \quad (5)$$

where V(r) is the intermolecular potential, r is equal to the distance between two atoms divided by the distance of the minimum energy (r_m), ε being the well depth and n is equal to m+gamma(r-1). The parameters for m, gamma and r_m and ε are shown below in table 2.3 [25].

Table 2.3: Parameters for GSM

GSM Potential Parameters				
Parameter	m	gamma	r_m (bohr)	ε (K)
Argon	13	7.5	7.10	141.55

The Aziz-Slaman potential is a little more involved than the previous two. The form of the potential can be observed in equations 6, 7 and 8.

$$V(r) = \varepsilon * A * \exp(-\alpha r + \beta r^2) - f(r)(\frac{C_6}{r^6} + \frac{C_8}{r^8} + \frac{C_{10}}{r^{10}}) \quad (6)$$

$$f(r) = \exp(-\left(\left(\frac{D}{r}\right) - 1\right)^2 \; if \; r < D \quad (7)$$

$$f(r) = 1 \; if \; r \geq D \quad (8)$$

where V(r) is the intermolecular potential, r is equal to the distance between two atoms divided by the distance of the minimum energy (r_m), ε is the well depth and α, β, C_6, C_8, C_{10}, D and A are all adjusted parameters. ε has units in Kelvin and r_m is in A.U. while the rest of the parameters are unitless. The Parameters can be observed in table 2.4 [26].

Table 2.4: Parameters for Aziz-Slaman

Aziz-Slaman Potential Parameters									
Parameter	**ε (K)**	**r_m(A.U.)**	**A**	**α**	**β**	**C_6**	**C_8**	**C_{10}**	**D**
Argon	143.224	7.10	99744.4	11.9196	-2.371	.651	3.686	-2.993	1.36

The AGY function is more involved than the previous three. The form of the potential can be observed below in equations 9, 10, 11 and 12.

$$V(r) = V_{HF}(R) + V_C(R) \quad (9)$$

$$V_{HF}(r) = Ae^{-\alpha R+\beta R^2} + \sum_{i=1}^{2} G_{1i}e^{-G_{2i}(R-G_{3i})^2} \quad (10)$$

$$V_C(R) = -(\sum_{i=1}^{2} G_{1i}e^{-G_{2i}(R-G_{3i})^2} + \sum_{n=3}^{8} f_{2n}(R,b)\frac{C_{2n}}{R^{2n}}) \quad (11)$$

$$f_{2n}(R,b) = 1 - \exp(-bR)\sum_{k=0}^{2n}\frac{(bR)^k}{k!} \quad (12)$$

where V(R) is the intermolecular potential, R is equal to the distance between two atoms, $V_{HF}(R)$ is the repulsive term, $V_C(R)$ is the attractive term and f_{2n}(R,b) is the damping function, and C_{2n} are the dispersion coefficients. The rest of the parameters can be observed below in table 2.5 [29].

Table 2.5: Parameters for AGY

AGY Potential Parameters		
Parameter	**Neon**	**Argon**
G_{11} (E_h)	-3.205E-04	2.1887E-04
G_{21} (a_0^{-1})	0.82172	0.13189
G_{31} (a_0)	3.2949	3.2949
G_{12} (E_h)	-2.771E-03	-1.4886E-02
G_{22} (a_0^{-1})	1.1097	0.46024
G_{32} (a_0)	2.2072	1.7768
A (E_h)	88.5513	82.9493
α (a_0^{-1})	2.20626	1.45485
β (a_0^{-2})	-0.0249851	-0.0379929
b (a_0^{-1})	1.85166	1.62365
C_6 ($E_h a_0^6$)	6.28174	63.752
C_8 ($E_h a_0^8$)	90.0503	1556.46
C_{10} ($E_h a_0^{10}$)	1679.45	4.944E+04
C_{12} ($E_h a_0^{12}$)	4.190E+04	2.073E+06
C_{14} ($E_h a_0^{14}$)	1.363E+06	1.105E+08
C_{16} ($E_h a_0^{16}$)	5.629E+07	7.248E+09

Figure 2.1 displays the two body potential functions of Argon and shows that all four functions line up very closely with the Aziz-Slaman having the lowest minimum. The energy minimum in all four cases is about 7.10 bohr. All the functions go from attractive to repulsive between 6.2 and 6.5 bohr.

The next two body function, which is available only for neon is the Korona/TT potential, and it is the most involved. Its form can be shown in Equation 13.

$$V(R) = Aexp(-\alpha R + \beta R^2) - \sum_{n=3}^{5} f_{2n}(R,b) * \frac{C_{2n}}{r^{2n}} \quad (13)$$

where R is the distances between the atoms, C_{2n} are the dispersion coefficients and A, α, β, and b are adjustable parameters. Then lastly, f_{2n} is the damping function which was shown above in equation 12. The first term is from Korona, and represents the repulsive term. The parameters can be observed in table 2.6 [27].

Table 2.6: Parameters for Korona/TT

Korona/TT Potential Parameters							
Parameter	A (A.U)	α ($bohr^{-1}$)	β ($bohr^{-2}$)	C_6 ($E_h bohr^6$)	C_8 ($E_h bohr^8$)	C_{10} ($E_h bohr^{10}$)	b ($bohr^1$)
Neon	78.52	2.133	-.035	6.96	49.87	2393.96	1.88

The last two body function is the modified TT potential. Its repulsive term is modified compared to the original TT. This can be observed in equation 14.

$$V(R) = Aexp(-a_1 R + a_2 R^2 + a_{-1} R^{-1} + a_{-2} R^{-2}) - \sum_{n=3}^{8} f_{2n}(R,b) * \frac{C_{2n}}{R^{2n}} \quad (14)$$

The parameters for the function for neon can be observed in table 2.7 [28].

Figure 2.2 displays the two body potential functions of neon and shows that all functions line up very closely and the modified TT has the lowest minimum. The energy minimums are about 5.92 bohr and both functions go from attractive to repulsive between 5.2 and 5.5 bohr.

Three Body Potentials

To investigate the behavior of our two benchmark methods we need a three body energy function for the rare gas solid. The three body energy potential functions used were Axilrod-Teller (for argon and helium) [3,4,20], Cohen-Murrell (for helium only) [5,6], Cencek (for helium and argon) [7,36], Extended Axilrod-Teller (for neon only), Modified Axilrod Teller (for neon only) and the Three Body potential of Ermakova (for neon only).

Table 2.7: Parameters for Modified TT

Modified TT Potential Parameters		
Parameter	**Unit**	**Neon**
A	K	4.02915058383 E7
$\mathbf{a_1}$	nm^{-1}	-4.28654039586 E1
$\mathbf{a_2}$	nm^{-2}	-3.33818674327
$\mathbf{a_{-1}}$	nm	-5.34644860719 E-2
$\mathbf{a_{-2}}$	nm^{2}	5.01774999419 E-3
b	nm^{-1}	4.92438731676 E1
$\mathbf{C_6}$	K nm^{6}	4.40676750157 E-2
$\mathbf{C_8}$	K nm^{8}	1.6489257701 E-3
$\mathbf{C_{10}}$	K nm^{10}	7.90473640524 E-5
$\mathbf{C_{12}}$	K nm^{12}	4.85489170103 E-6
$\mathbf{C_{14}}$	K nm^{14}	3.82012334054 E-7
$\mathbf{C_{16}}$	K nm^{16}	3.85106552963 E-8

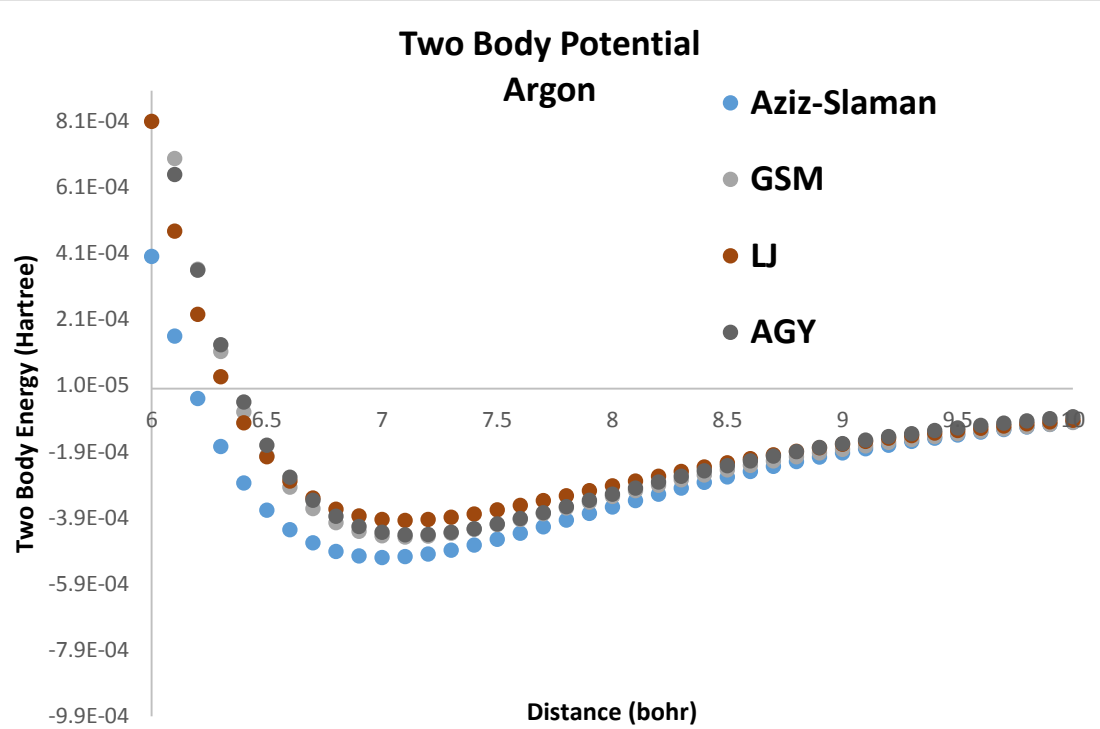

Figure 2.1: Two Body Potentials for Argon

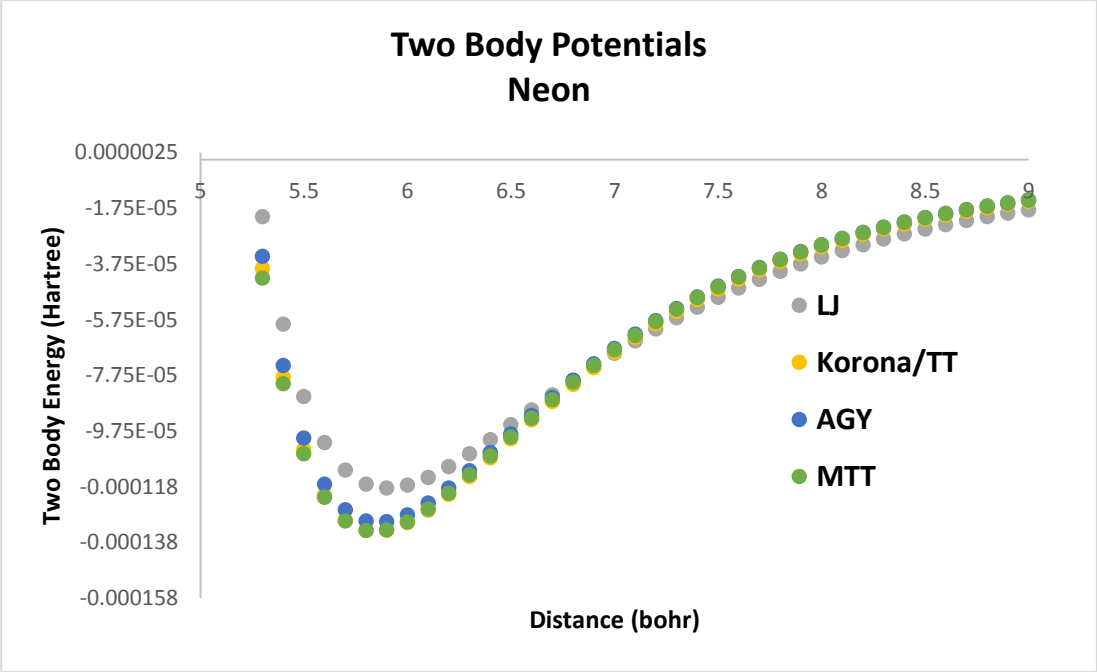

Figure 2.2: Two Body Potentials for Neon

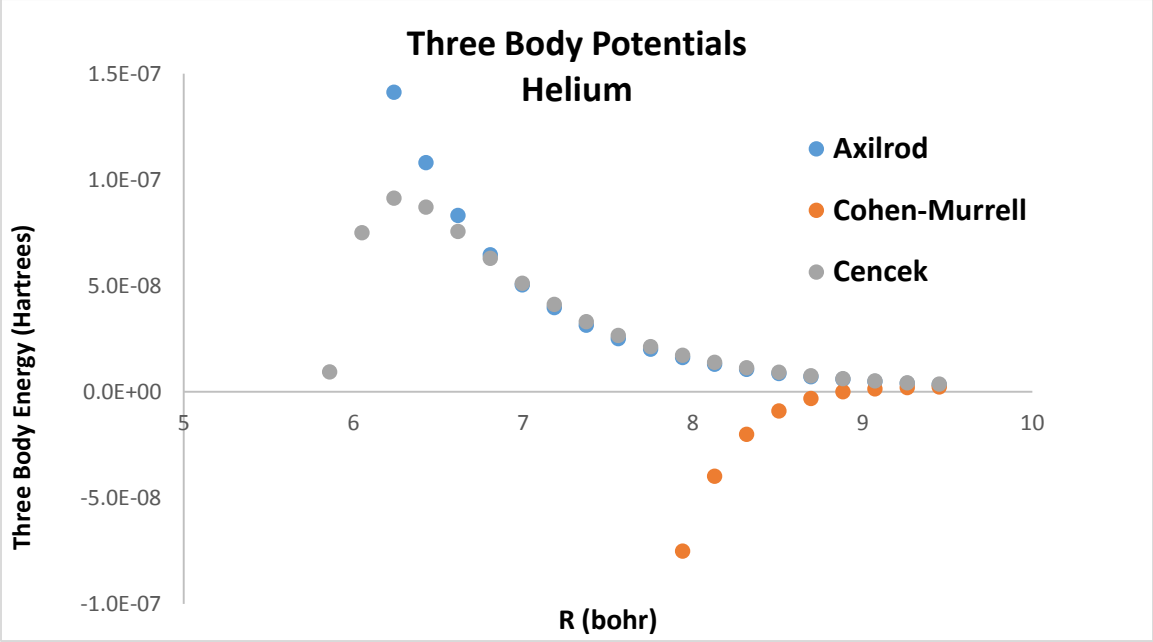

Figure 2.3: Three Body Potentials for Helium

The Axilrod-Teller function, also known as the triple dipole interaction, is the most universal as it can be applied to all systems. This function gives the three body dispersion energy, which is an analogue of the 2-body London dispersion energy that depends on $1/R^6$. The dispersion interactions arise from the fluctuating dipoles of the atoms or molecules. The formula for Axilrod-Teller can be observed below.

$$E3 = E0 * \frac{1+3\cos(a)*\cos(b)*\cos(c)}{(r_1 r_2 r_3)^3} \quad (15)$$

In the Axilrod-Teller function, variables r_1, r_2 and r_3 are the lengths of the sides of the triangle, a, b and c are the angles of the triangle, and E0 is the dispersion constant for the rare gas which has the value 1.47 E_h a_o^9 [6] for helium and 518.0 E_h a_o^9 for argon. The function shows that the Axilrod-Teller energy will be repulsive for equilateral triangular arrangements.

However, one of the drawbacks to the Axilrod-Teller model is that it makes the assumption that the electron clouds of the atoms do not overlap. Therefore, when two or three atoms get close the Axilrod-Teller function deviates from the true three body energy. This is addressed in the energy potential function developed by Cohen and Murrell. The Cohen-Murrell function of the helium three body interaction is much more involved than the Axilrod-Teller function, and it is based on quantum chemical calculations of the three body interactions. This function sets out to address the atomic overlap issues that make the Axilrod-Teller function inaccurate at small interatomic distances. The Axilrod-Teller function is incorporated into this function, but with a damping prefactor. The damping function is designed to include effects related to the atomic overlap of helium in small triangular configurations, since the Axilrod-Teller function assumes that the energy is dominated by dispersion forces. However, this is only true when the atoms do not overlap. Therefore, as the triangles get bigger the Cohen-Murrell function approaches the Axilrod-Teller function. The overall Cohen-Murrell function can be observed below.

$$E3 = V\{3\} + H(r1,r2,r3)V\{D\} \quad (16)$$

This incorporates Axilrod-Teller as V {D}. In the Cohen-Murrell function, V {3} is the exchange potential that includes the effect of atomic overlap and H(r1,r2,r3) is a product of the three damping functions for the three different sides of the triangle, so H(r1,r2,r3)=H(r1)H(r2)H(r3). The H(r) function will be 1 at large r and decreases as r decreases, so as the triangle becomes smaller the functions damp the Axilrod-Teller contribution. Although the Cohen-Murrell function is based on quantum chemical calculations, only calculations for isosceles triangles were utilized in calibrating the function. Detailed formulas for H(r) and V{3} can be found in reference [6].

The next function used was from Cencek and co-workers [7,36] for helium and argon, and it is also based on quantum chemical calculations. Unlike Cohen-Murrell, Cencek and co-workers used quantum chemical calculations for both isosceles and scalene helium triangles for building the function. Another benefit is that Cencek and co-workers performed much higher level quantum chemical calculations than Cohen-Murrell did, including a correction for the basis set superposition error. While this method is supposed to be the most accurate of the three body functions used for helium in this research, it is also the most time consuming to evaluate. The functional form of Cencek is much more complicated than either Axilrod-Teller or Cohen-Murrell. This is really what leads to much longer computing times. The behavior of the three functions for helium is shown in Figure 2.3 for an equilateral triangle of three helium atoms.

Figure 2.3 shows that for helium, the Cohen-Murrell function begins to differ from zero at about a nearest neighbor distance of 9 bohr, while the Axilrod-Teller and Cencek functions become nonzero, but with opposite signs, at about a nearest neighbor distance of about 5.9 bohr. This is important because these nearest neighbor distances roughly correspond to the densities that will be considered in this research. This can also be observed for argon in figure 2.4 as the Axilrod-Teller and Cencek functions start to differ with opposite signs at around 6.3 bohr [36].

The next three body potential function used was an Extended Axilrod-Teller function for neon. This function is similar to the original Axilrod-Teller; however, it includes parameters for the effects of electron cloud overlap. The format of the function can be observed below.

$$E3 = f\theta\left[c_0 r_g^{-9} + \left(c_1 + c_2 r_g^2 + c_3 r_g^4\right) e^{-c_4 r_s}\right] \quad (17)$$

where $f\theta$=(1+cos(a)cos(b)cos(c)), $r_g=(r_1r_2r_3)^{1/3}$, $r_s=r_1+r_2+r_3$ and the other components are adjusted parameters in atomic units, which can be shown in the table 2.8 [31].

The Modified Axilrod-Teller function was utilized next for neon. This function expands on the Extended Axilrod-Teller function; its form can be observed below in equation 18.

$$E3 = \left(-A\exp\left(-\alpha_{tr}(r_1 + r_2 + r_3)\right) + \frac{C_{tr}}{r_1^3 r_2^3 r_3^3}\right) * (1 + 3\cos(a)\cos(b)\cos(c)) \quad (18)$$

where the exponential term is the exchange three-body interaction, and the rest is the Axilrod-Teller function. The parameters can all be observed below in table 2.9 [32].

The last three body function employed was the Three Body potential of Ermakova (TBE). The form of this equation is counterpoise-corrected, and it can be observed below in equation 19.

$$E3 = \frac{a_1}{p^4(1+a_2p+a_3p^2)} + \frac{Q(a_4+a_5p)}{p^4(1+a_6p+a_7p^2)} + \frac{Q^2a_8}{p^3} \quad (19)$$

where p is the perimeter, Q is equal to cos(a)cos(b)cos(c) and the adjusted parameters can be observed below in table 2.10 [33]. The nature of the three body potentials for neon can be observed in figure 2.5 as the function hits the maximum at around 5.5 bohr and becomes attractive at around 5.1 bohr.

The two body and three body functions were based off empirical or quantum chemical calculations. The LJ, Aziz-Slaman, GSM, EAT, MAT, Cohen-Murrell and Axilrod-Teller potentials were based on empirical methods. The AGY, Korona/TT, MTT and Cencek were all based on Quantum chemical methods.

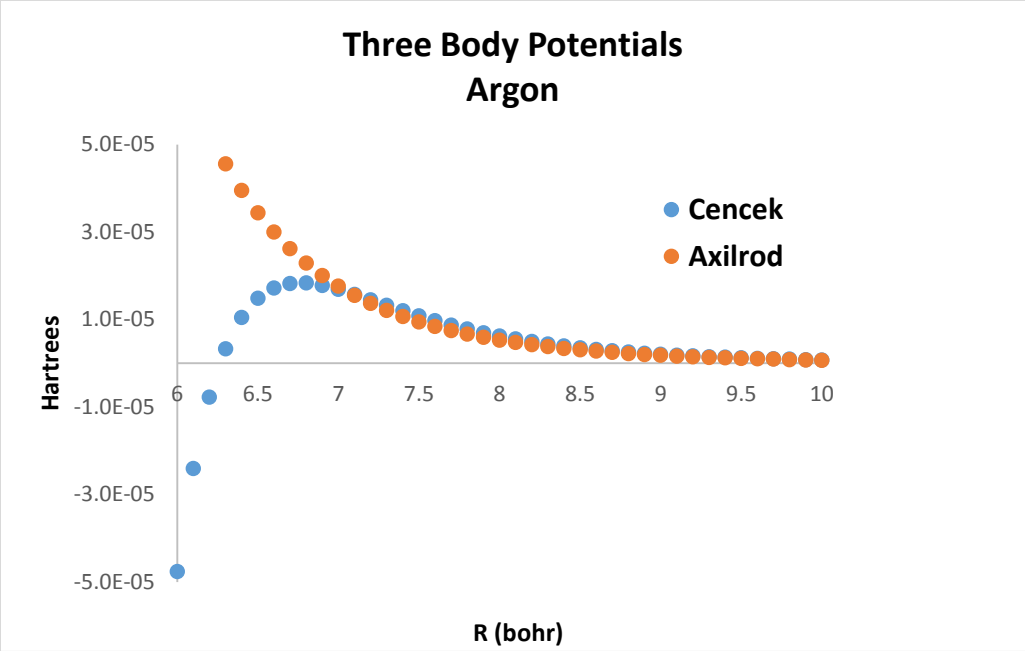

Figure 2.4: Three Body Potentials for Argon

Table 2.8: Parameters for Extended Axilrod-Teller

Extended Axilrod-Teller Potential Parameters					
Parameter	**C_0 (A.U)**	**C_1 (A.U.)**	**C_2 (A.U.)**	**C_3 (A.U.)**	**C_4 (A.U.)**
Neon	12.9236	466.449	-168.680	4.32545	1.23818

Table 2.9: Parameters for Modified Axilrod-Teller

Modified Axilrod-Teller Potential Parameters			
Parameter	**A (A.U.)**	**α_{tr}(A.U.)**	**C_{tr}(A.U.)**
Neon	566.969	1.1896	11.835

Table 2.10: Parameters for Three Body Potential of Ermakova

Three Body Potential of Ermakova Parameters								
Parameter	**a_1 (A.U.)**	**a_2 (A.U.)**	**a_3 (A.U.)**	**a_4 (A.U.)**	**a_5 (A.U.)**	**a_6 (A.U.)**	**a_7 (A.U.)**	**a_8 (A.U.)**
Neon	-1494.04	-2.40E-2	1.98E-4	-3.35E-4	196.84	-1.0E-2	4.64E-5	-77.46

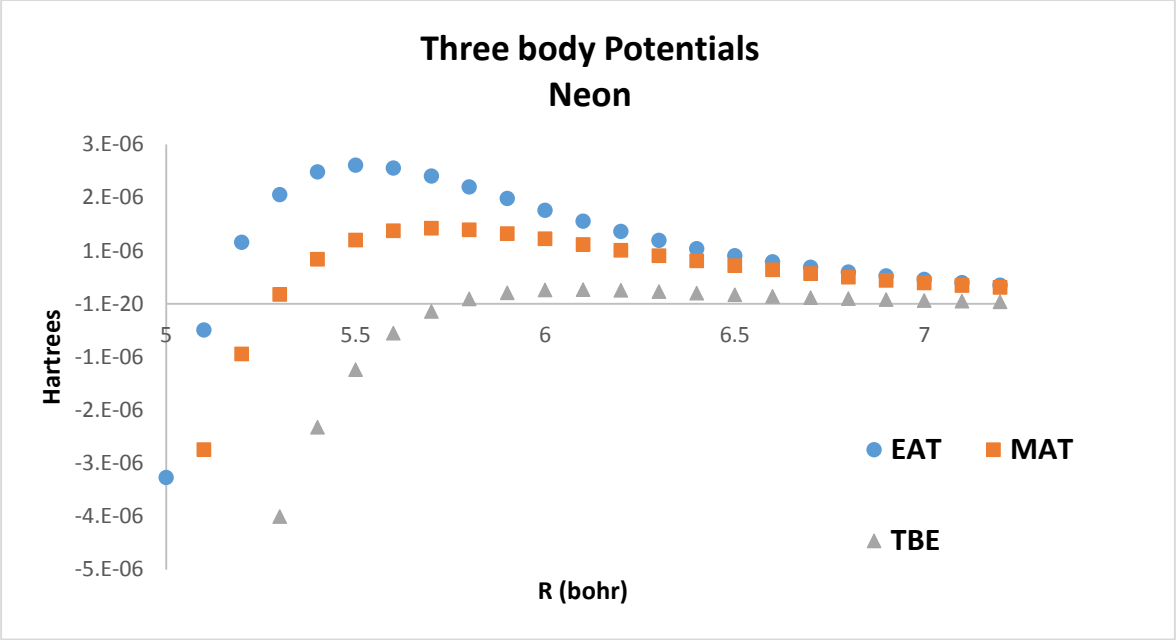

Figure 2.5: Three Body Potentials for Neon

Chapter 3: Methods

Accurate Methods

Gaussian Quadrature

If the atomic distributions are Gaussians, the natural first approach to calculating three-body energies including zero-point motion would be utilization of Gaussian quadrature. The evaluation of the integral of $e^{-x\wedge 2}$ multiplied by the function f(x) can be approximated using Gaussian quadrature by the following summation:

$$\int_{-\infty}^{+\infty} e^{-x^2} f(x)\, dx \approx \sum_{i=1}^{n} w_i f(x_i) \qquad (20)$$

where n is the number of quadrature nodes, x_i are the abscissas or positions of the nodes and w_i are their weights. However, if we consider the three-dimensional zero point motions of a set of three atoms, an abscissa or position is needed for each Cartesian coordinate of each atom. This yields a nine dimensional integral for evaluation. So overall the number of points or cycles can be written as n^9. The n values of n= 3, 4, 5, 6 and 7 were used for all triangles. With more nodes the number of cycles the computer must complete rises exponentially as n^9. For example, 20 nodes would correspond to 512 billion cycles that the computer will have to go through. This means that time for the calculation also goes up exponentially. The advantage to this method is smooth convergence, if the underlying function behaves normally.

In our application, however, Gaussian quadrature sometimes suffers from node overlap. This can be observed in Figure 3.1 and 3.2, which show the probability distribution for two nearest-neighbor atoms in solid helium according to the Einstein model with alpha=0.973 bohr and R=6.906 bohr. As more nodes are evaluated the set of locations of a given helium atom starts to expand further from the atom's average lattice site. Figure 3.1 shows for n=3 nodes the green and black markers as the possible locations of two neighboring helium atoms. The nodes are closely localized to the average lattice site. Figure 3.2 is n=7 nodes and shows that the locations of the helium atoms have clearly expanded and the outermost nodes are essentially on top each other. These outside locations here are weighted with a very low weight, but they are guaranteed to be evaluated as well during the Gaussian quadrature process. At low density, due to a larger mean squared displacement, node overlap becomes a serious problem for Gaussian quadrature evaluation of the three body energy for solid helium. Fortunately, at high density both of these issues start to become less important. Figure 3.3 clearly shows that the two Gaussian distributions overlap at low density. At high density this issue becomes minimal since the Gaussian distributions start to become narrower.

Monte Carlo Integration

The second approach to evaluating the three body energy is Monte Carlo integration. Monte Carlo integration uses a random point generator to estimate the value of multidimensional integrals.

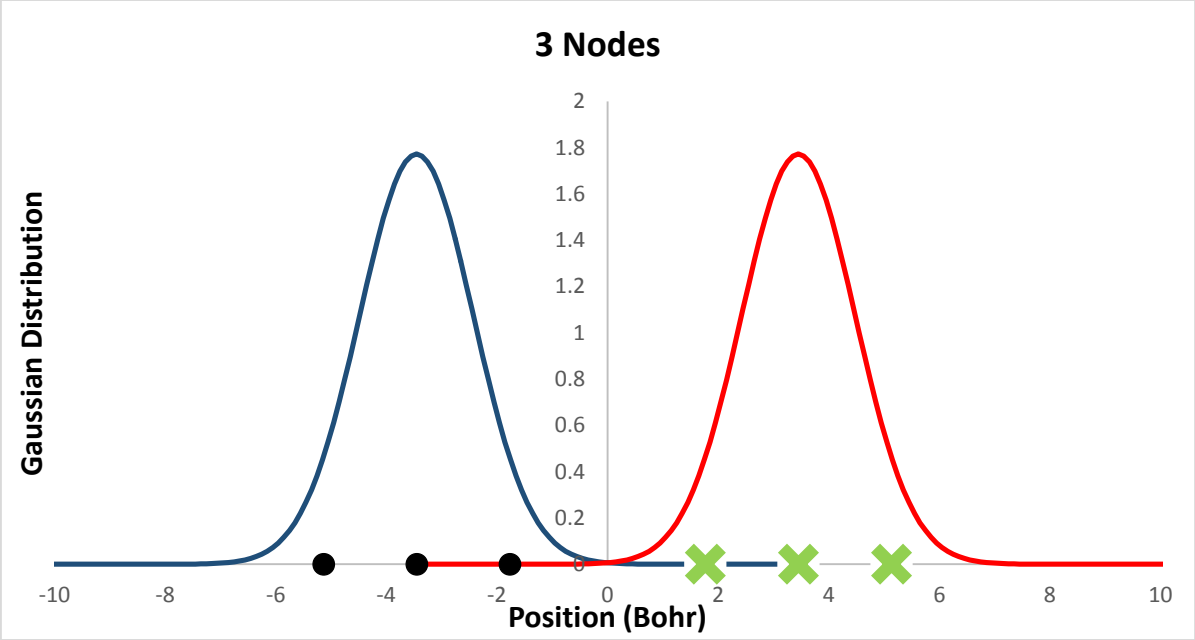

Figure 3.1: Position of three nodes for Gaussian Quadrature

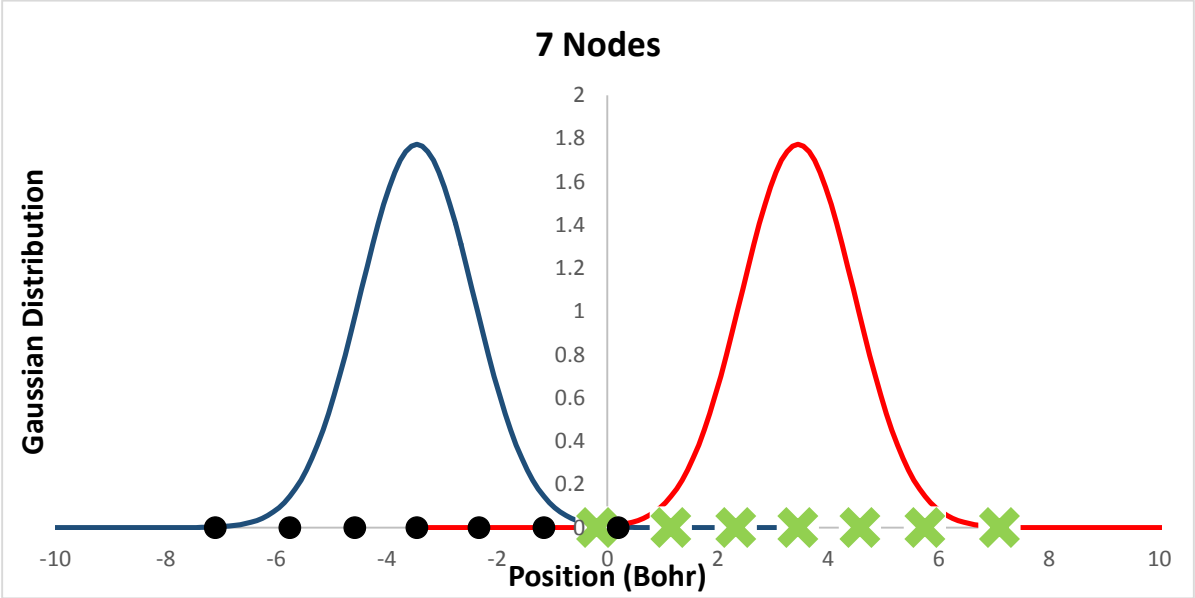

Figure 3.2: Position of seven nodes for Gaussian Quadrature

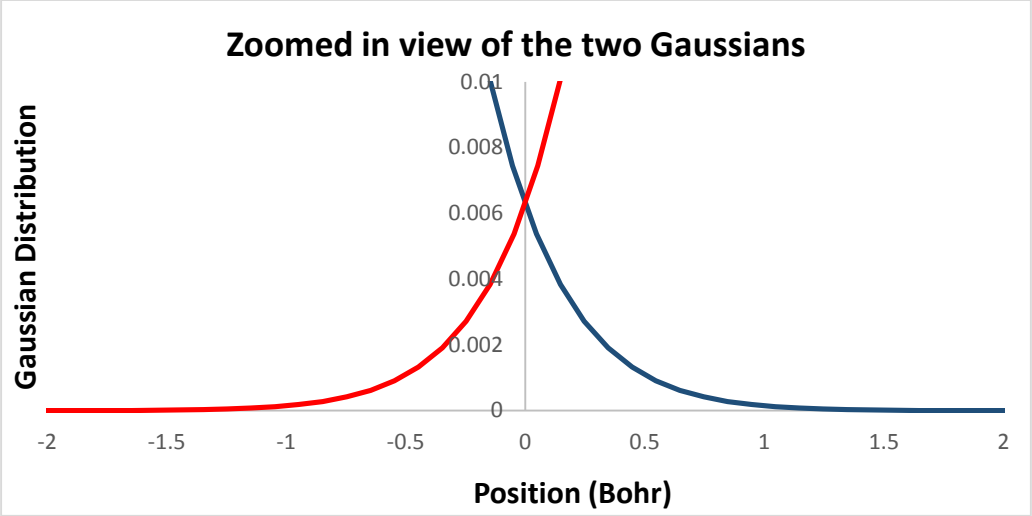

Figure 3.3: Gaussian overlap

This project used a pseudorandom number generator called Tausworthe88 [15,16] that generates values that are uniformly distributed between zero and one. In order to apply the random number generator to the Gaussian distribution the numbers from Tausworthe88 were converted according to the polar form of the Box-Muller transformation [22]:

$$Zo = \sqrt{-2\mathrm{Ln}(u1)} * \cos(2\pi * u2) \quad (21)$$

Here z0 is the random number associated with the Gaussian distribution and u1 and u2 are two random numbers from Tausworthe88. Repeating this process nine times generates the Cartesian coordinates of three atoms associated with the Gaussian distribution. Then to increase the accuracy of this method the number of cycles or points can be adjusted.

One big advantage to this method is that any number of points can be evaluated to compute an average three body energy. Hence, we are no longer locked in on a certain number of nodes or cycles like in Gaussian quadrature. Therefore, at low density it is much easier to use Monte Carlo over Gaussian quadrature. This is because Monte Carlo gives the flexibility to evaluate a rolling average and calculate confidence limits. This makes Monte Carlo the brute force method.

The downside to Monte Carlo, when applied to solid helium, is that while it is feasible to obtain an average three body energy at low density, the uncertainty is much greater than that of high density. Again this issue is significantly reduced at high density and doesn't show up for neon or argon. In terms of accuracy for helium both Gaussian quadrature and Monte Carlo integration depend heavily on the density that is being evaluated. For this work, the focus will be on two R values of 6.906 bohr (P≈ 28.6 Bar) and 6.553 bohr (P≈ 104.1 Bar). This will show key differences as $<U^2>$ becomes smaller. Also we will focus on the Cencek function since this function is expected to be the most accurate. At the R value of 6.906 bohr, the density will be lower, hence we will be dealing with a higher $<U^2>$ value. Therefore, as expected there are complications that arise with the large distribution of possible locations of the helium atoms. As you increase the number points or cycles Monte Carlo has fluctuation issues, which can be observed in Figure 3.4 and 3.5. The 95% confidence intervals represent degree of fluctuation.

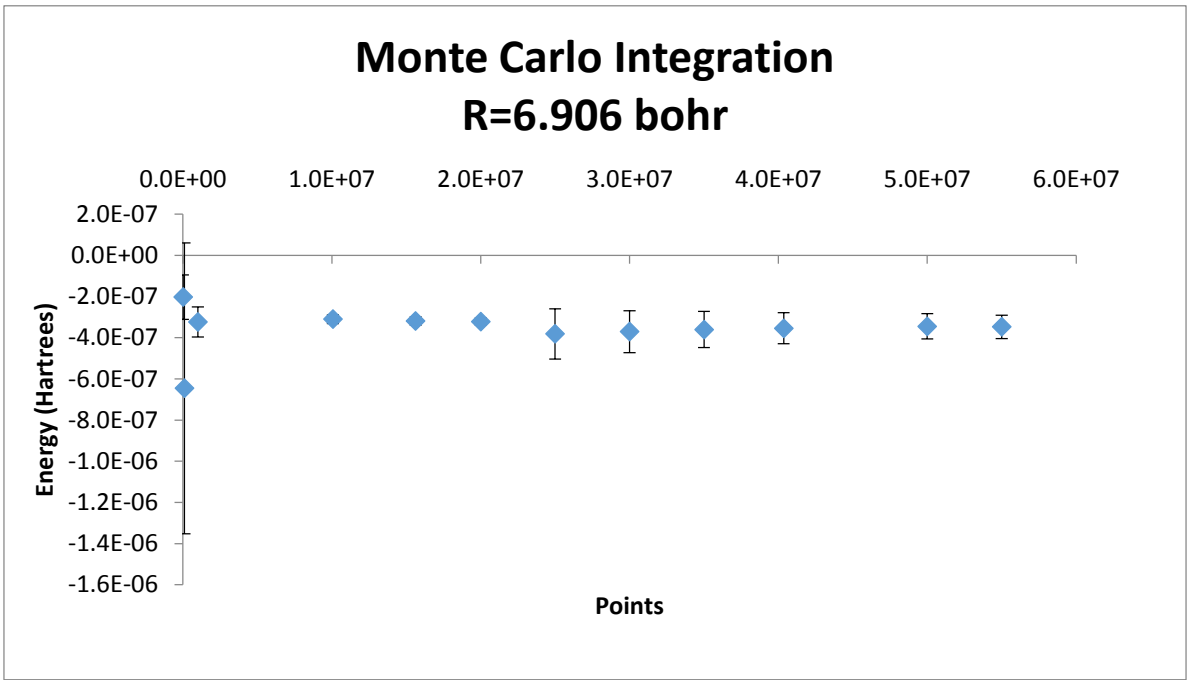

Figure 3.4: Monte Carlo Integration low density Helium (first 55 million points)

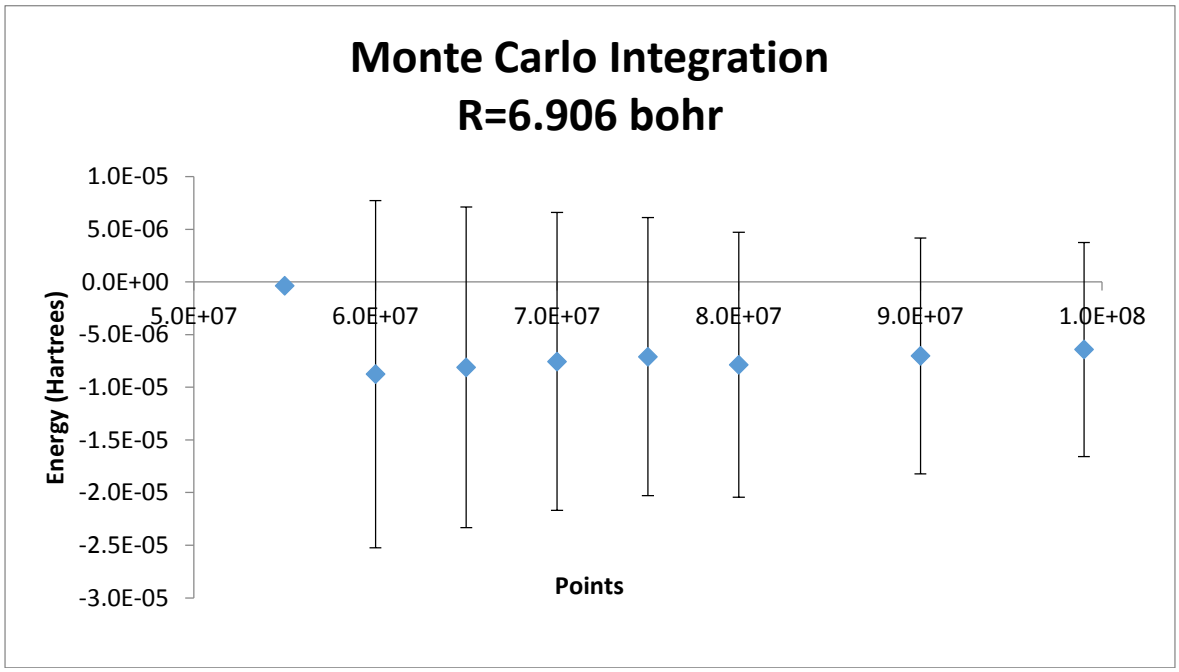

Figure 3.5: Monte Carlo Integration for low density Helium (next 45 million points)

At about 60 million points or cycles, the calculation hits a point where the error jumps along with the average. However, the advantage of Monte Carlo is that a rolling average can be obtained along with a calculation of a confidence interval. This allows a better idea or investigation of what is going on in the computation. This is why Monte Carlo has to be the brute force method, its flexibility just makes it a much more attractive method.

The next R value of 6.553 bohr shows a substantial improvement, as a slightly higher density means a lower $<U^2>$ value. This can be easily observed in Figure 3.6 and 3.7. In the figures, Monte Carlo clearly performs much better at a higher density. There still is fluctuation after 60 million points, however it is much smaller than at the R value of 6.906 bohr. Monte Carlo integration shows substantial improvement with high density as the confidence intervals are much narrower. These complications do not arise in neon and argon.

Approximate method

The approximate method used a polynomial approximation of the three body potential energy function. The simplest possible polynomial approximation is the one that results from considering each atom's motion along one of the three Cartesian directions away from the atom's lattice site. This will yield three trends in each direction for each atom totaling 9 in all. The polynomial approximation is therefore an estimate of the three body energy as the sum of nine polynomials, one for each of the Cartesian coordinates. Maple 18 [17] software was used to obtain the nine trends and graphs with X being the displacement and Y being the energies at that position. The following integral was then used to evaluate the zero-point motion's contribution to energy.

$$\frac{\int_{-\infty}^{\infty} \exp(-\frac{u^2}{2*\alpha^2})(polynomial\ approximation)du}{\int_{-\infty}^{\infty} \exp(-\frac{u^2}{2*\alpha^2})\,du} - (polynomial\ approximation\ intercept\ at\ u = 0) \quad (22)$$

where u is one of the nine variables in the polynomial approximation. The big benefit of this approach is speed, as it is the fastest method. However, its accuracy must be assessed.

Another way of representing the atomic motions, rather than observing them from a Cartesian point of view, is to use symmetry-based coordinates that describe the shape of a triangle formed by three atoms. One possible set of linear combinations can be observed below.

$$Q_1 = \frac{1}{\sqrt{3}}(r1 + r2 + r3) \quad (23)$$

$$Q_2 = \frac{1}{\sqrt{2}}(r2 - r3) \quad (24)$$

$$Q_3 = \frac{1}{\sqrt{6}}(2r1 - r2 - r3) \quad (25)$$

where Q1, Q2 and Q3 are the symmetry coordinates and r1, r2 and r3 are the sides of the triangles. So an advantage of the symmetry coordinates is that there are only three Q polynomials in comparison to nine Cartesian polynomials. Hence, equation 22 only needs to be computed three times for the Q's and summed up. This is more efficient compared to the Cartesian approach which needs to compute equation 22 nine times and sum them.

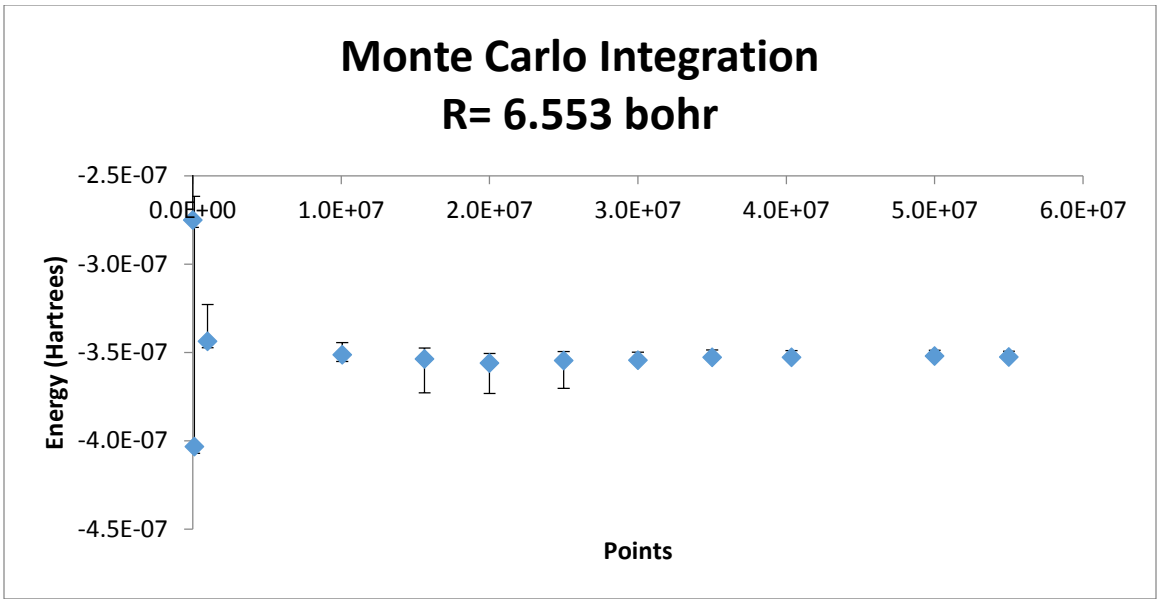

Figure 3.6: Monte Carlo Integration high density Helium (first 55 million points)

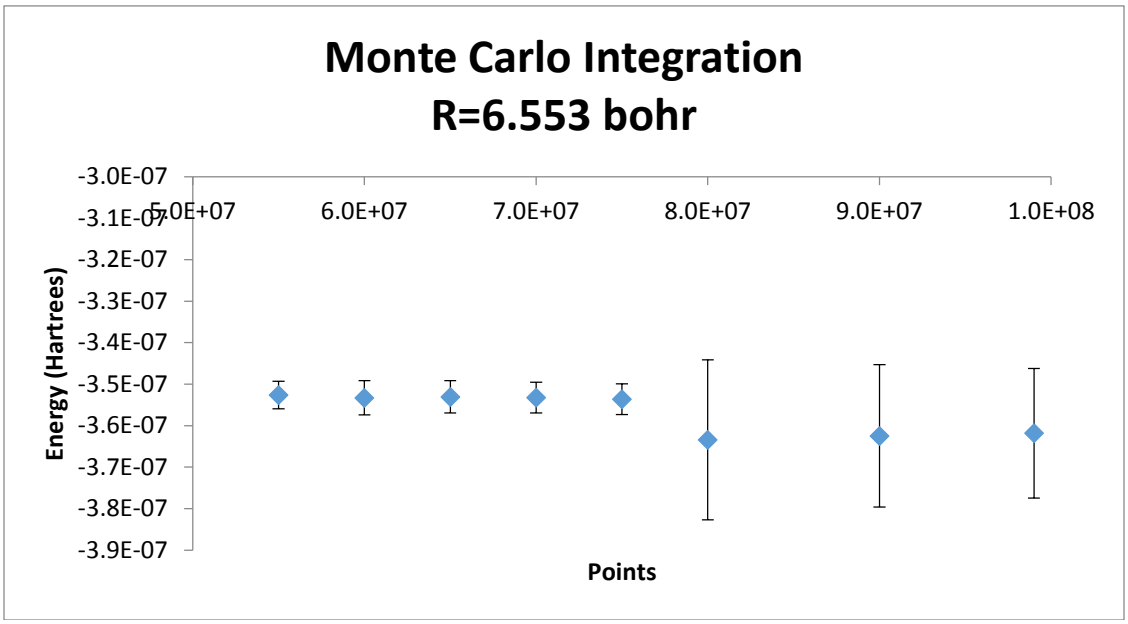

Figure 3.7: Monte Carlo Integration high density Helium (next 45 million points)

However, for this approach to be effective we must confirm that the distributions of three symmetry coordinates can accurately be represented by uncorrelated Gaussian distributions.

Symmetry Correlation

The correlation between distributions of Q1, Q2 and Q3 greatly depend on which triangle we are evaluating. The two triangles observed are the small equilateral triangle (SET) and small linear triangle (SLT). The SET is an equilateral triangle a nearest neighbor distance away from each atom. The SLT is a linear triangle which is just three atoms in a row separated by the nearest neighbor distance. Scatter plots were then employed in order to evaluate correlation. Starting with the SET, we can observe this for Q2 vs Q1, Q3 vs Q1 and Q3 vs Q2 in figures 3.8, 3.9 and 3.10 for solid helium with R=6.906 bohr and alpha=.973 bohr. The figures show that correlation is extremely small from the R^2 value for the SET. This implies that the Gaussian distributions are fairly uncorrelated for the SET. However, if we look at the SLT the correlation for Q2 vs Q1, Q3 vs Q1 and Q2 vs Q3 in figures 3.11, 3.12 and 3.13. In figure 3.13, Q3 vs Q1 yields an R^2 value of .9384. We can also take a look at the distribution of Q1, Q2 and Q3 to see how closely it resembles a Gaussian distribution. This can be seen in figures 3.14, 3.15, 3.16, 3.17, 3.18 and 3.19. The SET histograms all resemble Gaussian distributions. The SLT shows Q1 and Q2 resemble Gaussian distributions, where as Q3 appears to be a bit narrow. We can also take a look at Table 3.1 to observe kurtosis and skewness. It is shown that the SET has kurtosis values very close to 3.0 indicating a Gaussian distribution for helium, neon and argon. However, the SLT shows kurtosis values away from 3.0 indicating it does not match a Gaussian distribution well. Table 3.1 also shows that the distrbutions are slightly skewed as the values differ from zero. This confirms that adjustments to the TPA would need to be made when evaluating certain triangle configurations. This will be discussed in the future work section.

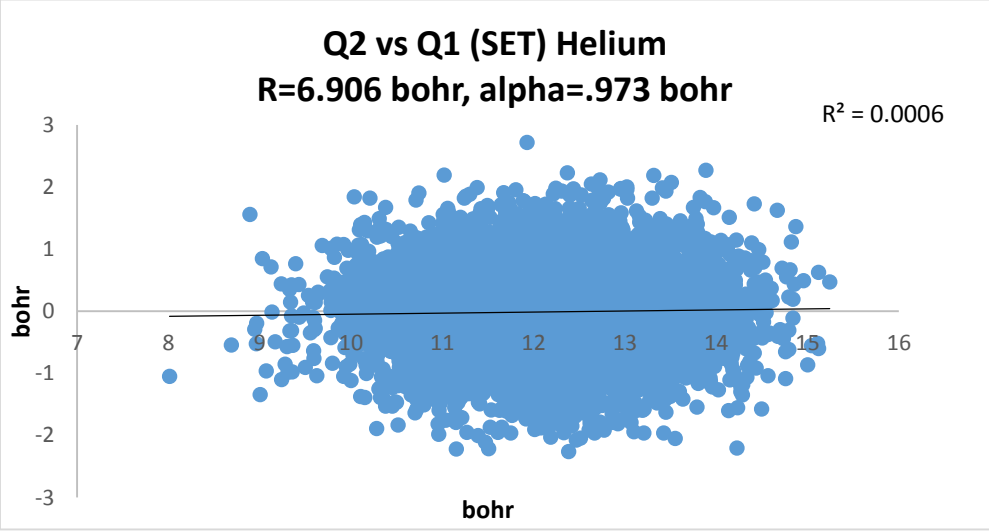

Figure 3.8: Q2 vs Q1 for SET

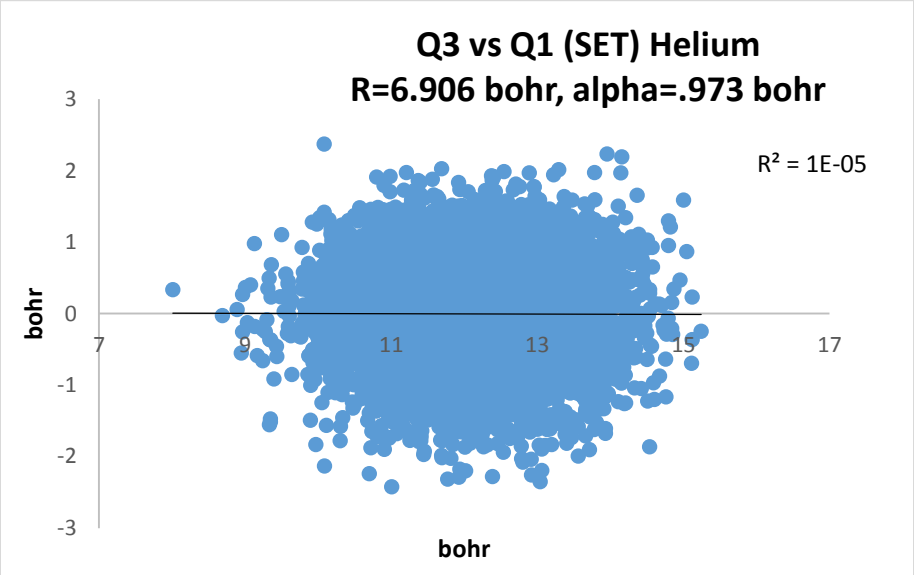

Figure 3.9: Q3 vs Q1 for SET

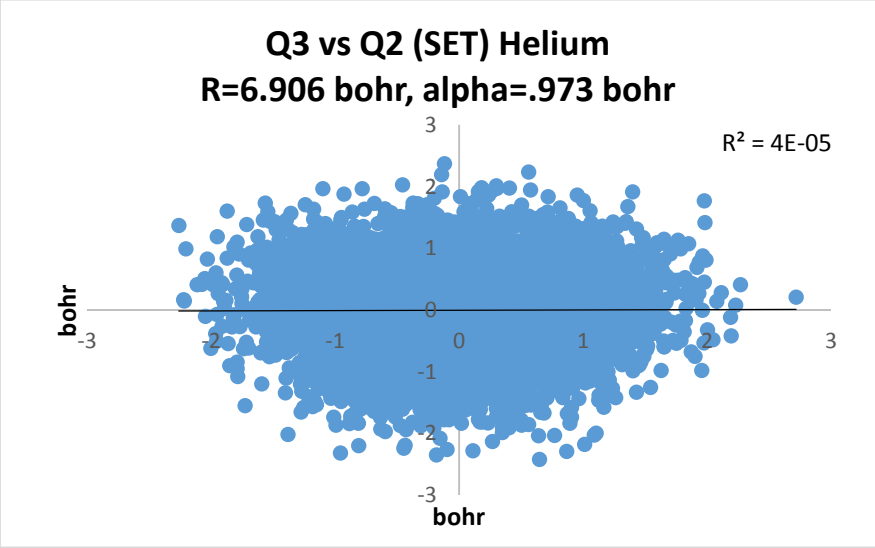

Figure 3.10: Q3 vs Q2 for SET

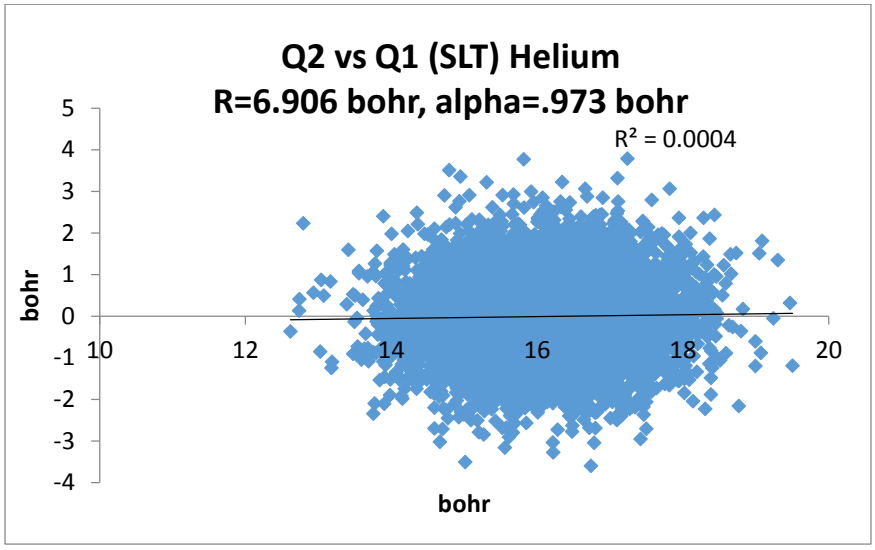

Figure 3.11: Q2 vs Q1 for SLT

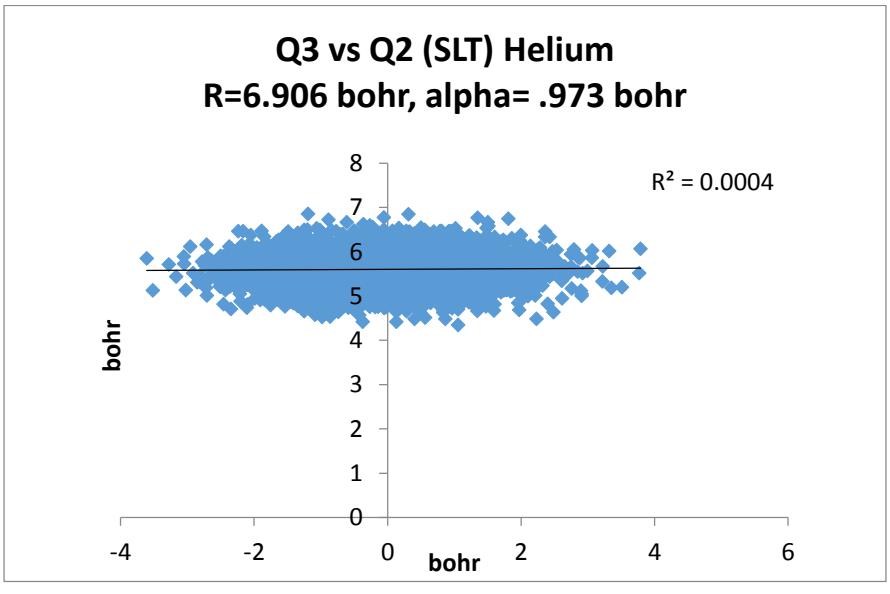

Figure 3.12: Q3 vs Q2 for SLT

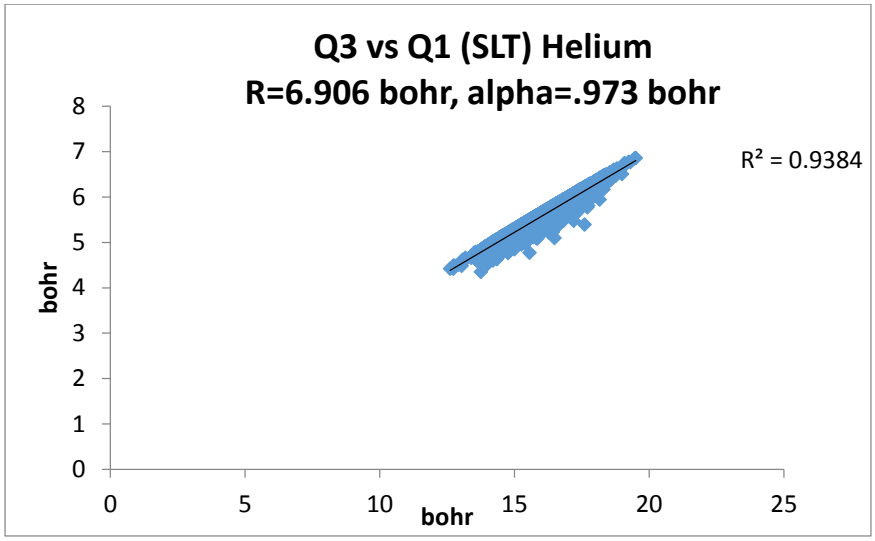

Figure 3.13: Q3 vs Q1 for SLT

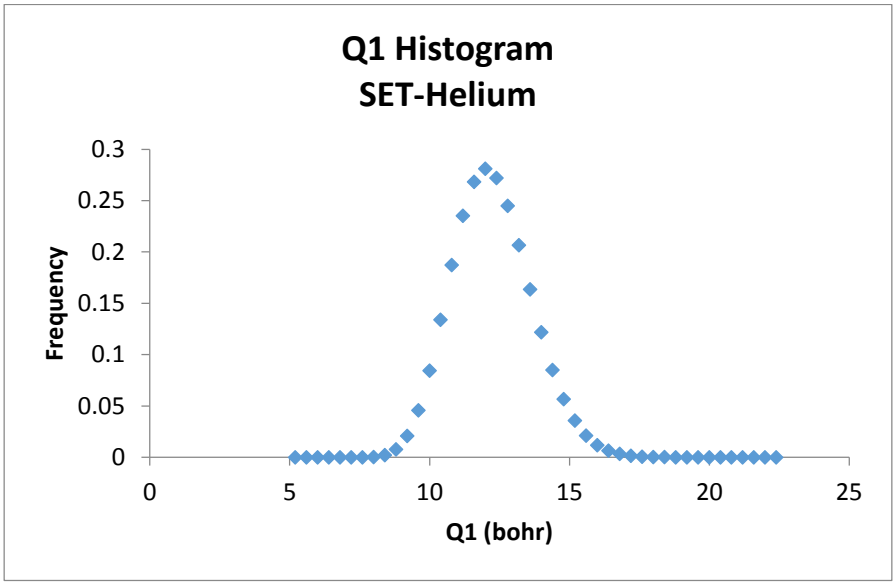

Figure 3.14: Q1 histogram for SET

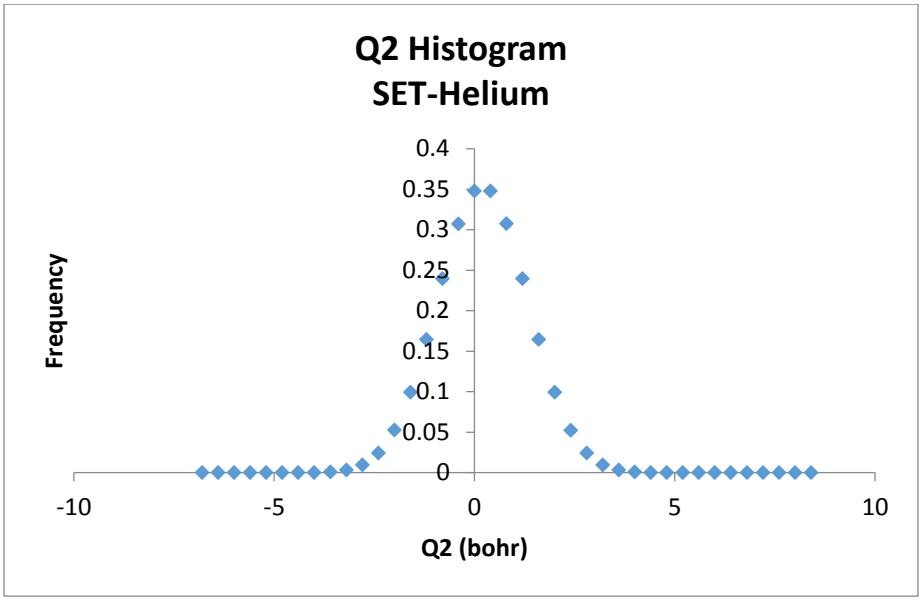

Figure 3.15: Q2 histogram for SET

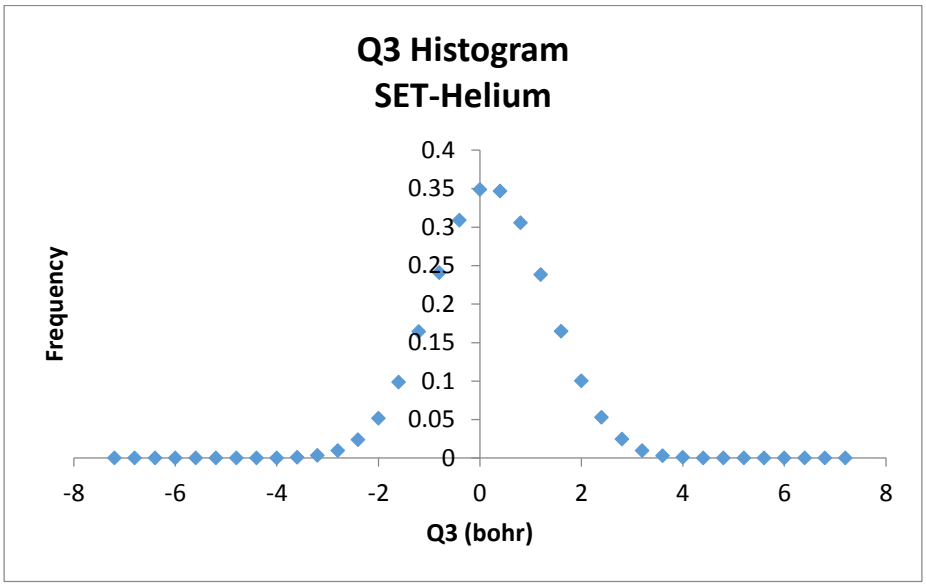

Figure 3.16: Q3 histogram for SET

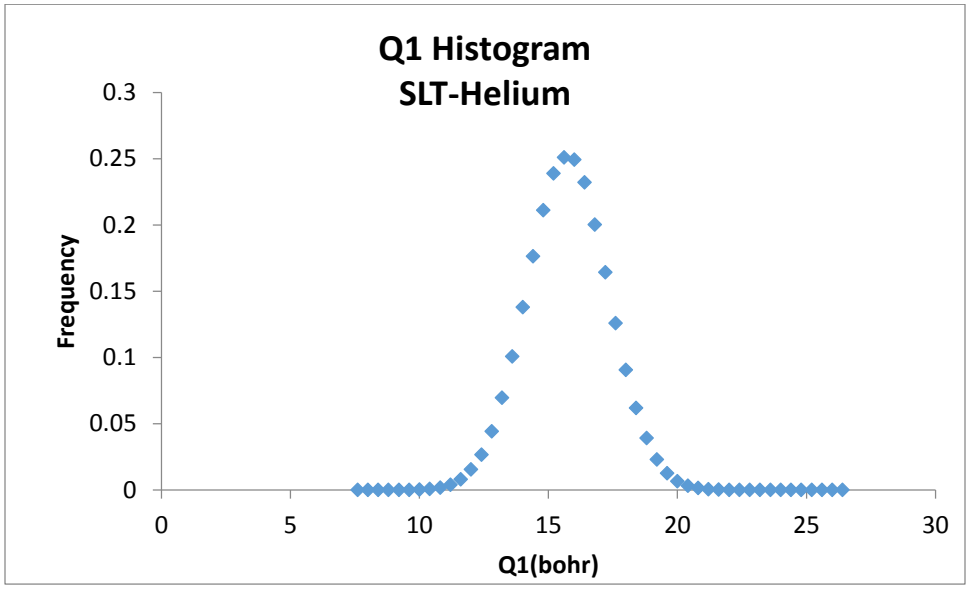

Figure 3.17: Q1 histogram for SLT

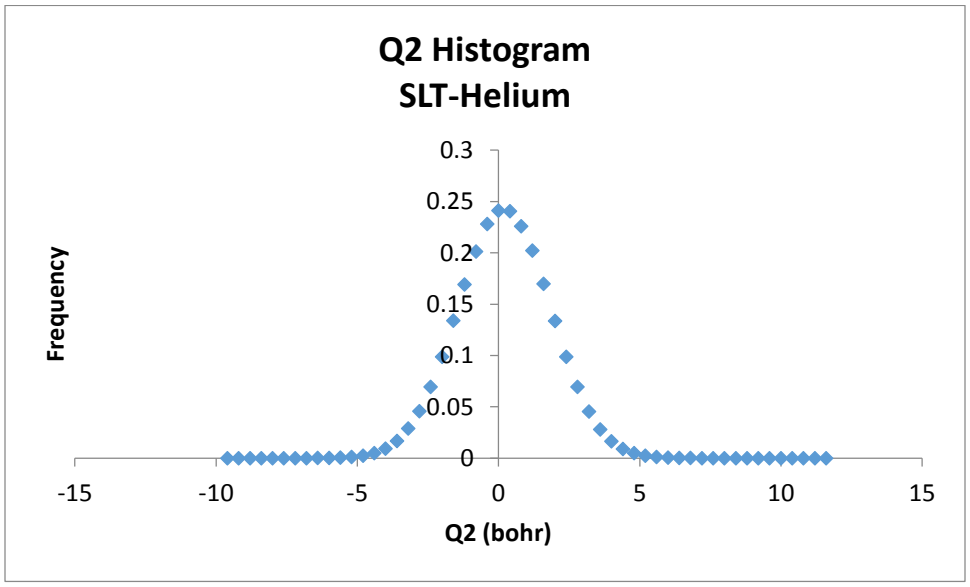

Figure 3.18: Q2 histogram for SLT

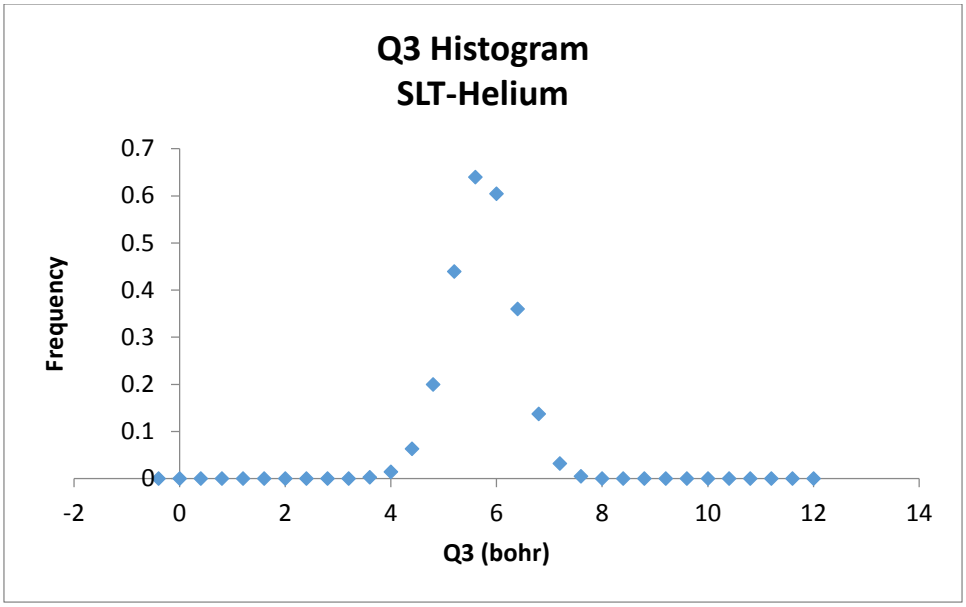

Figure 3.19: Q3 histogram for SLT

Triangle Configuration	Q	Helium Kurtosis	Neon Kurtosis	Argon Kurtosis	Helium Skewness	Neon Skewness	Argon Skewness
SET	**Q1**	2.99753	3.0020	3.0018	1.47E-02	1.83E-03	1.20E-03
SET	**Q2**	2.99531	3.0025	3.0027	2.22E-04	-4.36E-05	-9.70E-05
SET	**Q3**	2.99617	2.9971	2.9973	-1.97E-01	-7.74E-02	-4.08E-02
SLT	**Q1**	2.99668	3.0003	3.0001	5.27E-03	8.85E-04	6.88E-04
SLT	**Q2**	1.1541	1.0221	1.0062	-1.01E-03	8.77E-04	9.34E-04
SLT	**Q3**	1.11368	1.0171	1.0048	-5.14E-03	-8.71E-05	1.08E-04

Table 3.1: Kurtosis and Skewness for heium, neon and argon

Chapter 4: Results and Discussion

General Overview

In order to evaluate the Einstein model, comparisons will be made of cohesive energy. The cohesive energy consists of three components; kinetic energy, two body potential energy and three body potential energy, all of which need to be calculated. Equation 26 shows the relationship between cohesive energy and the three parts mentioned above. In this equation, the fractions 1/2 and 1/6 account for the different combinations of labeling pairs and triangles.

$$Cohesive\ Energy = KE + \frac{1}{2}\sum two\ body\ terms + \frac{1}{6}\sum three\ body\ terms \quad (26)$$

Each one of these parts will be addressed. Both the two body and three body terms will need to be summed up in the crystal. A visual representation can be observed below in Figure 4.1.

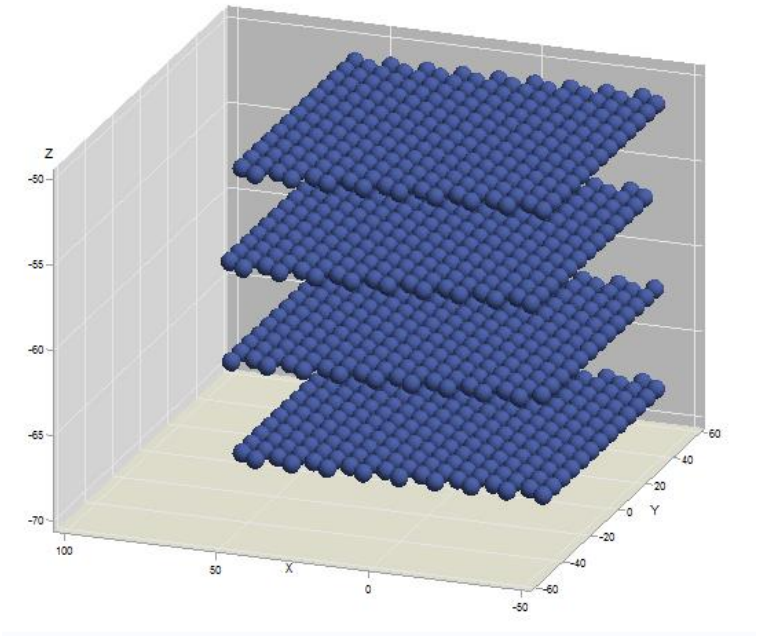

Figure 4.1 Four layers of an FCC crystal modeled

Figure 4.1 shows four layers of an FCC crystal. If we were to take the center atom as one of the two atoms in a pair and determine all pair wise combinations of this atom with another atom at a different lattice site, moving the second atom farther and farther away, until the energy contribution is insignificant, we would then account for all the two body contributions. The same thing is done for all the three body contributions. Then equation 26 will be computed when all three parts are evaluated. In the next several subsections, we show representative examples of the calculations of different components of the cohesive energy. We will then compare our calculations against experimental values. Finally, the chapter will be concluded with a discussion

of the challenges this approach faces when studying solid helium. Figures and tables for this section can be found in the appendix.

Two Body Contributions

The total two body energy contributions to the cohesive energy were calculated for both argon and neon, using the Lennard Jones potential as the two body energy function with parameters of sigma= 6.429 A.U. and epsilon= 3.699E-04 A.U. for argon and sigma=5.259 A.U. and epsilon= 1.180 E-04 A.U. for neon. The nearest neighbor distances were varied from a small value, representing a high density solid, to a value close to that for the diatomic energy minimum. The alpha value was then also varied to examine in a systematic way how the two body energy responds to both density and the Gaussian cloud size. All of the two body contributions are summed up to give the total two body contribution. These calculations were completed through MCI, with a typical statisical uncertainty of under 5%. The values can be observed in tables 4.1, 4.2, 4.3, 4.4 and 4.5 for both argon and neon. Also a visual representation of a surface graph can also be observed in figure 4.2 and 4.3 for argon and neon. The graphs for both neon and argon show the same general trend. The high density regions in figures 4.2 and 4.3 clearly show an upward energy trend. This would be because for both neon and argon, at high densities the nearest neighbor distances are in the repulsive region of the two body potential curve. As the densities become lower the two body contributions clearly display a downward energy trend as shown in figures 4.2 and 4.3. This is due to the nearest neighbor distances now being in the attractive region of the two body potential curve. The alpha value or Gaussian cloud size plays an important role as well. Regardless of the density as alpha is increased the two body energy contributions also increase. However, this effect is maximized at high density. This is clearly shown in figures 4.2 and 4.3 as there is a much steeper incline in energy as the alpha value is increased.

Three Body Contributions: Small Equilateral Triangles

The small equilateral triangle is the configuration formed by three pairs of nearest neighbor atoms in a crystal. This configuration is where three body energy is most important as it is where three atoms can come close in contact with each other simultaneously. This type of triangle shows up twenty-four times in the three body interactions that involve a specific atom in the crystal. Therefore, it is crucial to be able to calculate the three body energy of a small equilateral triangle accurately. Once again densities and alpha values are varied, and the three body energy of a small equilateral triangle is calculated. The Cencek function was used for argon and the Extended Axilrod-Teller was used for neon. These calculations were completed through MCI, with a typical statisical uncertainty of under 5%. Figures 4.4 and 4.5 show surface graphs for both neon and argon. The opposite trends can be observed compared to the two body trends. The graphs for neon and argon in figures 4.4 and 4.5 each show similar trends. At higher densities with small R values there is clearly a downward trend in three body energy. Additionally, as alpha is increased the three body energy becomes more negative, as shown from figures 4.4 and 4.5.

Three Body Contributions: Total

Total three body energy contributions were calculated for both argon and neon. The density and alpha values are varied again and the total three body energy contribution to the crystal is calculated. These calculations were completed through MCI, with a typical statisical uncertainty of under 5%. The values can be observed in tables 4.6, 4.7, 4.8, 4.9 and 4.10. A visual representation can also be observed in figures 4.6 and 4.7. The Cencek function was used for argon and the Extended Axilrod-Teller was used for neon. The same trends can be observed for the total three body energy contribution compared to the three body energy of a single small equilateral triangle. This is because the small equilateral triangle is the dominant contribution to the total three body energy. There is a clear downward trend as the density and alpha values increase. This is shown in figures 4.6 and 4.7.

Kinetic Energy

The kinetic energy can be evaluated using the ground state wave function of the harmonic oscillator. The harmonic oscillator's probability distribution function is a Gaussian which matches the assumption made in the Einstein model that the atomic motions are Gaussian. The form of the integral can be observed in equation 27 and 28.

$$KE = 3\int_{-\infty}^{\infty} {\psi_0}^* \left(-\frac{\hbar^2}{2\,m}\right)\frac{d^2}{dx^2}\psi_0\,dx \tag{27}$$

$$\psi_0 = \left(\frac{c}{\pi}\right)^{\frac{1}{4}} * e^{-\frac{cx^2}{2}} \tag{28}$$

where the factor of three accounts for the three X, Y and Z Cartesian directions and $\psi_0$is the ground state wave function for the harmonic oscillator. The computed values of the integral for both argon and neon can be observed in tables 4.11 and 4.12.

Cohesive Energy

In order to calculate the cohesive energy, the kinetic energy, two body contributions and three body contributions need to be summed up which can be observed in equation 26. Alternatively, most theoretical studies consider only two body contributions as observed in equation 29.

$$Cohesive\ Energy = KE + \frac{1}{2}\sum two\ body\ terms \tag{29}$$

Cohesive energies for only two body contributions and including three body contributions are shown for argon and neon in tables 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19, 4.20, 4.21 and 4.22. The Lennard Jones pair potential was used for argon and the Korona/TT pair potential was used for neon. The argon calculations used the Cencek three body potential and the neon calculations used the Extended Axilrod-Teller three body potential. The tables show that when three body contributions are added, there is a fairly sizable difference in the cohesive energy of the rare gas solids. This demonstrates that three body energies play a role and should be considered in an accurate model of the rare gas solids.

Comparison to Experimental

The cohesive energies computed using the Einstein model can be compared to the experimental values. We will evaluate our model at the nearest neighbor distances corresponding to the minimum energy values. The first thing to consider is what a realistic $<U^2>$ is for neon and argon at these densities. Lindemann parameters are provided by work done by Henry Glyde [8]. This can be observed in equations 30 and 31 for neon and argon.

$$.091 = \frac{\sqrt{<U^2>}}{R} \text{ for neon.} \tag{30}$$

$$.048 = \frac{\sqrt{<U^2>}}{R} \text{ for argon.} \tag{31}$$

This will yield an alpha value of .312 bohr for neon and .198 bohr for argon. There are also a number of two body functions and three body functions that can be evaluated. Therefore, the percent error can be observed in tables 4.23 for neon and argon. In the case of neon, the best agreement with experiment comes from the Korona/TT and TBE potentials, with 0.37% error. Argon has the best agreement with LJ and Cencek at 0.71% error and also AGY and Cencek at 0.73% error. Overall, depending on the two body and three body functions utilized the Einstein can be a very good approximation in systems like neon and argon.

TPA Evaluation

The next piece of the puzzle is to evaluate the polynomial approximation. There is no three body potential function yet available for molecular systems such as H_2 or methane. If the TPA method demonstrates good agreement with Monte Carlo, then it may be possible to evaluate three body energies in molecular systems without having to build a global three body energy function. The small equilateral triangle plays the most crucial role in three body energy. Therefore, we will compare the three body energy of a small equilateral triangle obtained from MCI against that obtained from TPA. Neon will be evaluated first. Some general trends can be observed as we vary alpha and the crystal density, as observed in tables 4.24 and 4.25.

In both table 4.24 and table 4.25, it is clearly shown that at a small alpha value there is better agreement, compared to when the cloud size or alpha value increases. This is true regardless of whether symmetry or Cartesian coordinates are utilized. We can now take a look at the agreement between the TPA and Monte Carlo, when Glyde's $<U^2>$ are implemented, which is shown in tables 4.26 and 4.27.

At neon's energy minimum the TPA gives fairly good agreement with Monte Carlo, especially with the symmetry coordinates. The TPA has the most trouble when the nearest neighbor distance for the solid is closest to the point where the three body function for the small equilateral triangle changes sign from positive to negative. This point is around 5.1 bohr for the EAT function. The R values of 4.8066 and 5.4134 bohr clearly have more trouble with agreement. Argon was evaluated next, and the same general trends are observed for solid argon, as we vary alpha and the density of the solid. The results are shown in tables 4.29 and 4.30.

Once again a clear trend can be observed as the higher densities appear to yield better agreement for both Cartesian and symmetry coordinates. Next we consider the use of the TPA method for solid neon at a density corresponding to the minimum energy value, which is a lower density. This can be seen in table 4.28 for symmetry and Cartesian coordinates.

As we saw before, a smaller alpha or cloud size yields better agreement. Once again, good agreement between the TPA approach and Monte Carlo integration can be observed, when Glyde's $<U^2>$ are used. This can be seen in tables 4.31 and 4.32.

For an equilateral triangle of three argon atoms, the Cencek three body energy function crosses from positive to negative at around R = 6.3 bohr. This plays a larger role in the TPA method using symmetry coordinates, as the Q1 coordinate relates to the perimeter of the equilateral triangle. Lastly, the TPA performance at the energy minimum for argon was evaluated in table 4.33.

At argon's energy minimum the TPA gives fairly good agreement with Monte Carlo, however, this time the Cartesian coordinates perform better. At this point, it can be concluded that the TPA is not at all perfect, but manageable. In the future work, it will be discussed on the improvements that can be made to the TPA.

Complications of Helium

As stated in the methods section, due to the high zero-point motion of solid helium there is a lot of fluctuation observed in the results obtained using Gaussian quadrature and Monte Carlo integration, making it difficult to pinpoint an accurate average three body energy. A comparison of the Monte Carlo results from the Einstein model to those using configurations provided by Barnes, is shown in figure 4.8. The Barnes configurations are based on quantum Monte Carlo simulations and include the effect of atomic correlation.

Figure 4.8 clearly shows that Monte Carlo integration performs very poorly for the Einstein model. This demonstrates for solid helium that pure Einstein behavior will not yield a good approximation to Monte Carlo. These problems continue when implementing the TPA. This can be observed as the density is varied. The two nearest-neighbor distances that will be used will be 6.906 bohr and 4.999 bohr (P≈ 8400 Bar). These R values were chosen to represent a low density case and high density case. At R=6.906 bohr ($<U^2>$=2.8398 bohr^2), the TPA approaches differ substantially from Monte Carlo results, regardless of the order of the polynomial expansion. This is shown in table 4.34.

Table 4.34 shows there are a lot of fluctuations in the TPA results as the order of the polynomial approximation changes, and this yields large percent differences when the TPA reults are compared against Monte Carlo results. On top of that polynomial approximation does not perform very well with a higher $<U^2>$ value. The Cartesian results tend to stay a little more stable, compared to the results based on symmetry coordinates. However, when a lower molar volume or higher density is observed it shows great improvement. This can be observed for the R value of 4.999 bohr ($<U^2>$= 0.4196 bohr^2) in table 4.35.

The Cartesian approach clearly performs better than the Q-based approach, but agreement with Monte Carlo integration is clearly better across the board than at the lower density. The two polynomial approximation calculations both stay fairly stable above the tenth order polynomial. This is a little more encouraging, as it shows how as the density increases the agreement becomes better. In closing, both methods need substantial improvements before they can be applied to helium, as neither one can really be deemed accurate. The improvements to these methods will be discussed in the future work section.

Equation of States

The cohesive energy data can be utilized to construct an equation of state. Using estimations of the alpha values from tables 4.15 and 4.16, we can contruct a graph of Cohesive Energy Vs Nearest Neighbor Distance. This graph can be observed in figure 4.9. This data could be used to calculate the pressure volume curves. In order to obtain more accurate cohesive energies, we would need to calculate energies using a finer grid of alpha values.

Chapter 5: Conclusion and Future Work

Neon and Argon

Our studies of both neon and argon have demonstrated that the Einstein model can be used to evaluate three body energies in systems with small zero-point fluctuations. The experimental cohesive energy for these systems agreed fairly well with the energy predicted by the Einstein model depending on the combination of two body and three body potentials used. For neon, the best agreement came from the TT two body potential and the TBE three body potential which yielded 0.37 % error. On the other hand, for argon the best agreement was observed using the Lennard Jones two body potential and Cencek three body potential at 0.71% error, or using the AGY two body function and the Cencek three body potential at 0.73% error.

The most important area for future work for neon and argon will be for improving the TPA. Results clearly show that there was room for improvement. The first area for improvement to make TPA a more accurate method is to investigate the role of cross terms. In the TPA, to obtain the polynomial, one variable changes as the others are held constant. For example, the Cartesian approach moves an atom in one direction holding the other eight coordinates constant to get each polynomial. The approach based on symmetry coordinates would vary one of the three symmetry coordinates (Q1, Q2, Q3) and keep the other two constant to get each polynomial. A cross term here would essentially take into account simultaneous changes in two or more coordinates which is probably a more realistic way to describe the properties of the three-body energy. This could give the symmetry coordinates an advantage over the Cartesian. On the surface it was shown that the Cartesian-based approach performs better than the approach using symmetry coordinates, however since there are only three variables for the symmetry coordinates there would be a lot fewer cross terms than if Cartesians were used. So while the Cartesian approach appears to be more accurate in some cases, the ease of adding cross terms to the symmetry coordinates could potentially outweigh the use of the Cartesians depending how much improvement is seen, especially since most of our test cases have less than 5 % error.

Something else to consider is that the definitions of the symmetry coordinates (Q1, Q2, Q3) were taken from reference [6], but these are only one possible set of symmetry coordinates for a three atom system. From investigating scatter plots of the three pairs of symmetry coordinates (Q1, Q2), (Q1, Q3) and (Q2, Q3), we believe that there may be some slight correlation between Q_2 and Q_3 for small equilateral triangle configurations. It may be beneficial to obtain a new definition of the symmetry coordinates with less correlation between the different symmetry coordinates.

Helium

Overall, for solid helium the calculations at high density clearly are more reliable than the calculations at low density. The large Gaussian distribution at low density creates complications that cause fluctuation and error. Therefore, using the Einstein model for helium would not be an accurate approximation. There need to be modifications to the Einstein model.

A possible area to improve the Einstein model is to explore the use of new atomic distribution functions that eliminate the spurious exchange of atoms between neighboring sites. This would control the location of the atoms and would allow us to correct for the overlapping shown in figure 3.3. Since the Einstein model is based off of a Gaussian distribution, altering the distribution might be an easy fix to get a huge improvement in the model. Once a new atomic distribution function is established then two body and three body energies can be evaluated which will allow us to compute the cohesive energy. Then the last piece of the puzzle would be to compare the TPA approach and Monte Carlo integration. Improvements to the TPA that were discussed above in context of solid neon and solid argon could also be implemented. If these improvements are successful, the TPA approach would serve as a bridge to estimate three body energies of other systems or molecules that do not have a full three body energy function. This would allow us to estimate three body contributions to the cohesive energy of molecular solids such as H_2, methane and benzene.

References

[1] Irikura, Karl K. "Experimental vibrational zero-point energies: Diatomic molecules." *Journal of physical and chemical reference data* 36, no. 2 (2007): 389-397.

[2] Zucker, I. J. "The Zero-Point Energy and Equation of State of Solid Helium at the Absolute Zero." *Proceedings of the Physical Society* 73, no. 6 (1959): 965.

[3] Sadus, Richard J. "Exact calculation of the effect of three-body Axilrod–Teller interactions on vapour–liquid phase coexistence." *Fluid Phase Equilibria* 144, no. 1 (1998): 351-359.

[4] von Lilienfeld, O. Anatole, and Alexandre Tkatchenko. "Two-and three-body interatomic dispersion energy contributions to binding in molecules and solids." *The Journal of chemical physics* 132, no. 23 (2010): 234109.

[5] Ujevic, Sebastian, and S. A. Vitiello. "Analysis of the contributions of three-body potentials in the equation of state of 4He." *The Journal of chemical physics* 119, no. 16 (2003): 8482-8491.

[6] Cohen, Michael J., and John N. Murrell. "An analytic function for the three-body potential of He_3." *Chemical physics letters* 260, no. 3 (1996): 371-376.

[7] Cencek, Wojciech, Konrad Patkowski, and Krzysztof Szalewicz. "Full-configuration-interaction calculation of three-body nonadditive contribution to helium interaction potential." *The Journal of chemical physics* 131, no. 6 (2009): 064105.

[8] Glyde, Henry. "Helium, Solid." University of Delaware. http://www.physics.udel.edu/~glyde/Solid_H13.pdf (accessed January 2014).

[9] Driessen, A., E. Van der Poll, and Isaac F. Silvera. "Equation of state of solid He 4." *Physical Review B* 33, no. 5 (1986): 3269.

[10] Arms, D. A., R. S. Shah, and R. O. Simmons. "X-ray Debye-Waller factor measurements of solid 3 He and 4 He." *Physical Review B* 67, no. 9 (2003): 094303.

[11] Fefferman, A. D., F. Souris, A. Haziot, J. R. Beamish, and S. Balibar. "Dislocation networks in 4 He crystals." *Physical Review B* 89, no. 1 (2014): 014105.

[12] Rościszewski, Krzysztof, and Beate Paulus. "The zero-point energy in the molecular hydrogen crystal." *Molecular Physics* 108, no. 16 (2010): 2147-2152.

[13] J.H. Noggle, Physical Chemistry (3rd edition), (1996: New York, Harper-Collins) adapted from M. Dole, Introduction to Statistical Thermodynamics, (1954: New York, Prentice-Hall).

[14] F. Seitz, *The Modern Theory of Solids* (McGraw-Hill Book Company, Inc., New York, 1940), pp 99-105.

[15] Pierre L'Ecuyer, Maximally equidistributed combined Tausworthe generators, *Mathematics of Computation*, v.65 n.213, p.203-213, Jan. 1996 [doi>10.1090/S0025-5718-96-00696-5]

[16] Generating Gaussian Random Numbers. http://www.design.caltech.edu/erik/Misc/Gaussian.html (accessed December 2014).\

[17] "Maple 18. Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario."

[18] Barnes, Ashleigh L., and Robert J. Hinde. "Search for anisotropy in the Debye-Waller factor of HCP solid 4He." *The Journal of Chemical Physics* 144, no. 8 (2016): 084505.

[19] Hinde, Robert J. "Three-body interactions in solid parahydrogen." *Chemical Physics Letters* 460, no. 1 (2008): 141-145.

[20] Axilrod, B. M.; Teller, E. "Interaction of the van der Waals Type Between Three Atoms". *Journal of Chemical Physics* vol. 11 p. 299 (1943).

[21] Herrero, Carlos P. "Solid helium at high pressure: a path-integral Monte Carlo simulation." *Journal of Physics: Condensed Matter* 18, no. 13 (2006): 3469.

[22] G. E. P. Box and Mervin E. Muller, "A Note on the Generation of Random Normal Deviates", *The Annals of Mathematical Statistics*, Vol. 29, p. 610 (1958).

[23] Cazorla, C., and J. Boronat. "Zero-temperature equation of state of solid 4He at low and high pressures." *Journal of Physics: Condensed Matter* 20, no. 1 (2008): 015223.

[24] Jones, John Edward. "On the determination of molecular fields. II. From the equation of state of a gas." In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 106, no. 738, pp. 463-477. The Royal Society, 1924.

[25] Bobetic, M. V., and J. A. Barker. "Solid-state properties of argon, krypton, and xenon near 0 K from an [n (r)]− 6 potential." *Physical Review B* 28, no. 12 (1983): 7317.

[26] Goharshadi, Elaheh K., and Mohsen Abbaspour. "Molecular dynamics simulation of argon, krypton, and xenon using two-body and three-body intermolecular potentials." *Journal of chemical theory and computation* 2, no. 4 (2006): 920-926.

[27] Nasrabad, A. Eskandari, R. Laghaei, and U. K. Deiters. "Prediction of the thermophysical properties of pure neon, pure argon, and the binary mixtures neon-argon and argon-krypton by Monte Carlo simulation using ab initio potentials." *The Journal of chemical physics* 121, no. 13 (2004): 6423-6434.

[28] Hellmann, Robert, Eckard Bich, and Eckhard Vogel. "Ab initio potential energy curve for the neon atom pair and thermophysical properties of the dilute neon gas. I. Neon–neon interatomic potential and rovibrational spectra." *Molecular Physics* 106, no. 1 (2008): 133-140.

[29] Giese, Timothy J., Vanessa M. Audette, and Darrin M. York. "Examination of the correlation energy and second virial coefficients from accurate ab initio calculations of rare-gas dimers." *The Journal of chemical physics* 119, no. 5 (2003): 2618-2622.

[30] Linderberg, Jan, and Fred W. Bystrand. "Cohesive energy of solid neon." *Arkiv Fysik* 26 (1964).

[31] Schwerdtfeger, P., and Andreas Hermann. "Equation of state for solid neon from quantum theory." *Physical Review B* 80, no. 6 (2009): 064106.

[32] Freiman, Yu A., and S. M. Tretyak. "Many-body interactions and high-pressure equations of state in rare-gas solids." *Low Temperature Physics* 33, no. 6 (2007): 545-552.

[33] Ermakova, Elena, Jan Solca, Gerold Steinebrunner, and Hanspeter Huber. "Ab Initio Calculation of a Three-Body Potential To Be Applied in Simulations of Fluid Neon." *Chemistry–A European Journal* 4, no. 3 (1998): 377-382.

[34] Silvera, Isaac F. "The solid molecular hydrogens in the condensed phase: Fundamentals and static properties." *Reviews of Modern Physics* 52, no. 2 (1980): 393.

[35] R.K. Crawford, in *Rare Gas Solids*, edited by M. L. Klein and J. A. Venables (Academic, New Yourk, 1976), Vol 2, Chap. 11.

[36] W. Cencek, G. Garberoglio, A.H. Harvey, M.O. McLinden and K. Szalewicz 2013. Three-body nonadditive potential for argon with estimated uncertainties and third virial coefficient. *The Journal of Physical Chemistry A*, *117*(32), pp.7542-7552.

Appendix

Table 4.1: Total two body contribution for Neon 1

Two Body Potential Energy- Neon All Two-Body Energies in Hartrees R values (row) in bohr Alpha values (columns) in bohr						
	3.989	**4.179**	**4.353**	**4.515**	**4.665**	**4.807**
0.201	1.05E-01	6.19E-02	3.73E-02	2.26E-02	1.36E-02	7.92E-03
0.220	1.09E-01	6.49E-02	3.93E-02	2.40E-02	1.46E-02	8.60E-03
0.238	1.14E-01	6.81E-02	4.14E-02	2.54E-02	1.56E-02	9.31E-03
0.254	1.19E-01	7.14E-02	4.36E-02	2.69E-02	1.66E-02	1.00E-02
0.270	1.24E-01	7.47E-02	4.59E-02	2.85E-02	1.77E-02	1.08E-02
0.284	1.30E-01	7.82E-02	4.82E-02	3.01E-02	1.88E-02	1.16E-02
0.298	1.35E-01	8.19E-02	5.07E-02	3.18E-02	2.00E-02	1.25E-02
0.312	1.41E-01	8.56E-02	5.32E-02	3.35E-02	2.12E-02	1.33E-02
0.324	1.47E-01	8.95E-02	5.58E-02	3.53E-02	2.25E-02	1.42E-02
0.337	1.53E-01	9.35E-02	5.85E-02	3.72E-02	2.38E-02	1.52E-02
0.348	1.60E-01	9.77E-02	6.13E-02	3.91E-02	2.52E-02	1.62E-02

Table 4.2: Total two body contribution for Neon 2

Two Body Potential Energy- Neon All Two-Body Energies in Hartrees R values (row) in bohr Alpha values (columns) in bohr						
	4.940	**5.067**	**5.188**	**5.303**	**5.413**	**5.520**
0.201	4.30E-03	1.97E-03	4.63E-04	-5.07E-04	-1.12E-03	-1.50E-03
0.220	4.79E-03	2.33E-03	7.24E-04	-3.14E-04	-9.79E-04	-1.40E-03
0.238	5.30E-03	2.70E-03	9.98E-04	-1.10E-04	-8.27E-04	-1.28E-03
0.254	5.83E-03	3.09E-03	1.28E-03	1.03E-04	-6.67E-04	-1.16E-03
0.270	6.39E-03	3.49E-03	1.58E-03	3.27E-04	-4.99E-04	-1.03E-03
0.284	6.97E-03	3.92E-03	1.90E-03	5.62E-04	-3.22E-04	-9.00E-04
0.298	7.58E-03	4.36E-03	2.23E-03	8.09E-04	-1.36E-04	-7.59E-04
0.312	8.21E-03	4.83E-03	2.57E-03	1.07E-03	5.86E-05	-6.11E-04
0.324	8.87E-03	5.31E-03	2.93E-03	1.34E-03	2.63E-04	-4.56E-04
0.337	9.56E-03	5.82E-03	3.31E-03	1.62E-03	4.78E-04	-2.92E-04
0.348	1.03E-02	6.35E-03	3.71E-03	1.92E-03	7.03E-04	-1.20E-04

Table 4.3: Total two body contribution for Neon 3

Two Body Potential Energy- Neon All Two-Body Energies in Hartrees R values (row) in bohr Alpha values (columns) in bohr						
	5.622	**5.700**	**5.800**	**5.900**	**5.920**	**6.000**
0.201	-1.73E-03	-1.83E-03	-1.89E-03	-1.89E-03	-1.89E-03	-1.86E-03
0.220	-1.65E-03	-1.76E-03	-1.84E-03	-1.86E-03	-1.85E-03	-1.83E-03
0.238	-1.56E-03	-1.69E-03	-1.79E-03	-1.82E-03	-1.82E-03	-1.80E-03
0.254	-1.47E-03	-1.62E-03	-1.73E-03	-1.78E-03	-1.78E-03	-1.77E-03
0.270	-1.37E-03	-1.54E-03	-1.67E-03	-1.73E-03	-1.74E-03	-1.74E-03
0.284	-1.27E-03	-1.46E-03	-1.61E-03	-1.69E-03	-1.69E-03	-1.71E-03
0.298	-1.16E-03	-1.37E-03	-1.55E-03	-1.64E-03	-1.65E-03	-1.67E-03
0.312	-1.05E-03	-1.28E-03	-1.48E-03	-1.58E-03	-1.60E-03	-1.63E-03
0.324	-9.31E-04	-1.18E-03	-1.40E-03	-1.53E-03	-1.55E-03	-1.59E-03
0.337	-8.05E-04	-1.08E-03	-1.32E-03	-1.47E-03	-1.49E-03	-1.54E-03
0.348	-6.73E-04	-9.75E-04	-1.24E-03	-1.41E-03	-1.43E-03	-1.50E-03

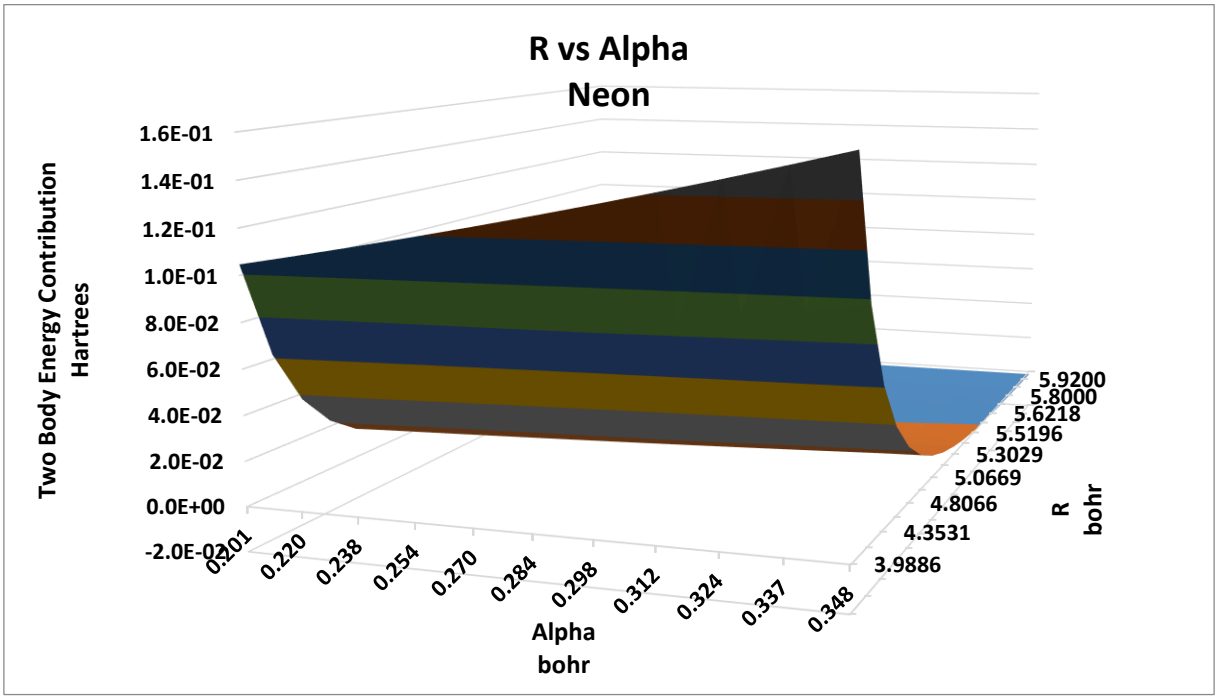

Figure 4.2: Surface graph of total two body contribution for R vs Alpha (Neon)

Table 4.4: Total two body contribution for Argon 1

Two Body Potential Energy- Argon All Two-Body Energies in Hartrees R values (row) in bohr Alpha values (columns) in bohr							
	5.0253	**5.2650**	**5.4846**	**5.6879**	**5.8777**	**6.0559**	**6.2242**
0.134	2.90E-01	1.46E-01	7.59E-02	3.95E-02	1.96E-02	8.38E-03	1.88E-03
0.146	2.97E-01	1.50E-01	7.79E-02	4.07E-02	2.04E-02	8.84E-03	2.17E-03
0.158	3.05E-01	1.53E-01	8.00E-02	4.19E-02	2.11E-02	9.30E-03	2.48E-03
0.169	3.12E-01	1.57E-01	8.21E-02	4.31E-02	2.18E-02	9.78E-03	2.79E-03
0.179	3.20E-01	1.61E-01	8.43E-02	4.44E-02	2.26E-02	1.03E-02	3.10E-03
0.189	3.29E-01	1.65E-01	8.66E-02	4.57E-02	2.34E-02	1.08E-02	3.43E-03
0.198	3.37E-01	1.70E-01	8.89E-02	4.70E-02	2.42E-02	1.13E-02	3.76E-03
0.207	3.46E-01	1.74E-01	9.13E-02	4.84E-02	2.50E-02	1.18E-02	4.10E-03
0.216	3.56E-01	1.79E-01	9.37E-02	4.98E-02	2.59E-02	1.23E-02	4.45E-03
0.224	3.65E-01	1.83E-01	9.62E-02	5.13E-02	2.68E-02	1.29E-02	4.80E-03
0.231	3.75E-01	1.88E-01	9.89E-02	5.28E-02	2.76E-02	1.34E-02	5.17E-03

Table 4.5: Total two body contribution for Argon 2

Two Body Potential Energy- Argon All Two-Body Energies in Hartrees R values (row) in bohr Alpha values (columns) in bohr							
	6.3839	**6.5359**	**6.6812**	**6.8204**	**6.9542**	**7.0830**	**7.1070**
0.134	-1.92E-03	-4.12E-03	-5.35E-03	-5.99E-03	-6.27E-03	-6.32E-03	-6.32E-03
0.146	-1.72E-03	-3.98E-03	-5.25E-03	-5.93E-03	-6.22E-03	-6.29E-03	-6.28E-03
0.158	-1.52E-03	-3.84E-03	-5.16E-03	-5.86E-03	-6.17E-03	-6.25E-03	-6.25E-03
0.169	-1.31E-03	-3.70E-03	-5.06E-03	-5.79E-03	-6.12E-03	-6.22E-03	-6.22E-03
0.179	-1.10E-03	-3.55E-03	-4.96E-03	-5.71E-03	-6.07E-03	-6.18E-03	-6.18E-03
0.189	-8.79E-04	-3.40E-03	-4.85E-03	-5.64E-03	-6.02E-03	-6.14E-03	-6.15E-03
0.198	-6.57E-04	-3.25E-03	-4.75E-03	-5.57E-03	-5.97E-03	-6.11E-03	-6.11E-03
0.207	-4.30E-04	-3.10E-03	-4.64E-03	-5.49E-03	-5.91E-03	-6.07E-03	-6.07E-03
0.216	-1.98E-04	-2.94E-03	-4.53E-03	-5.41E-03	-5.86E-03	-6.03E-03	-6.04E-03
0.224	4.02E-05	-2.77E-03	-4.42E-03	-5.33E-03	-5.80E-03	-5.99E-03	-6.00E-03
0.231	2.84E-04	-2.61E-03	-4.30E-03	-5.25E-03	-5.74E-03	-5.94E-03	-5.96E-03

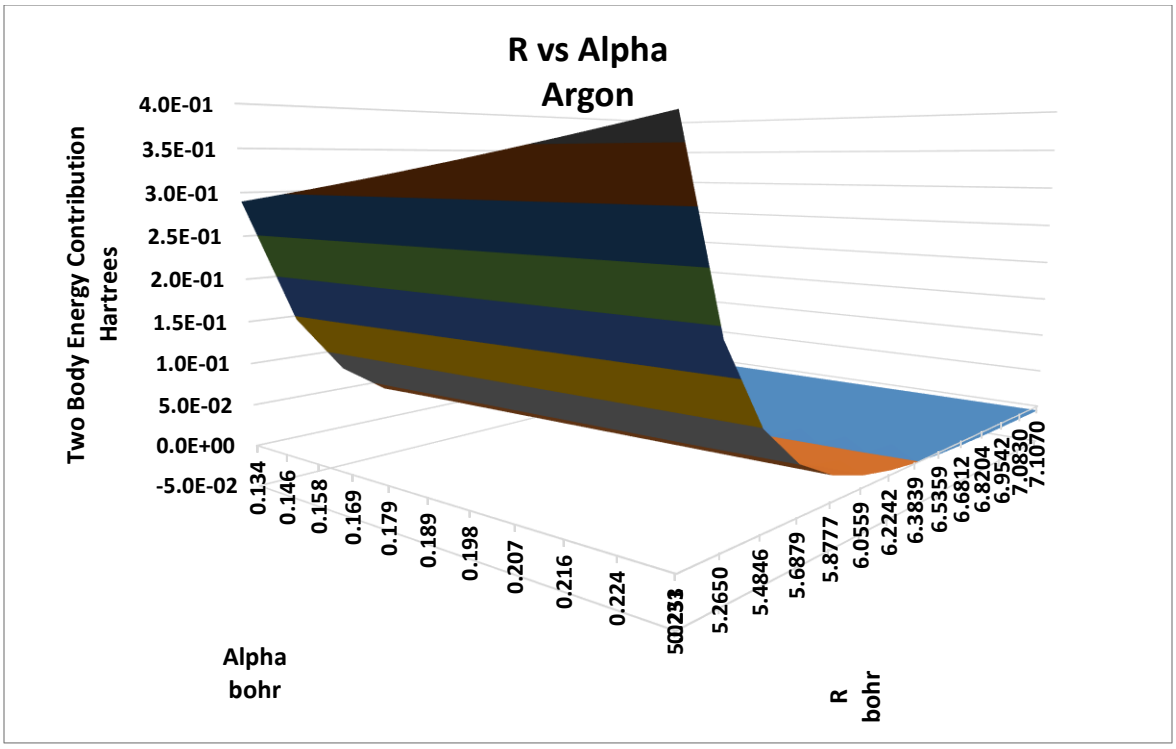

Figure 4.3: Surface graph of total two body contribution for R vs Alpha (Argon)

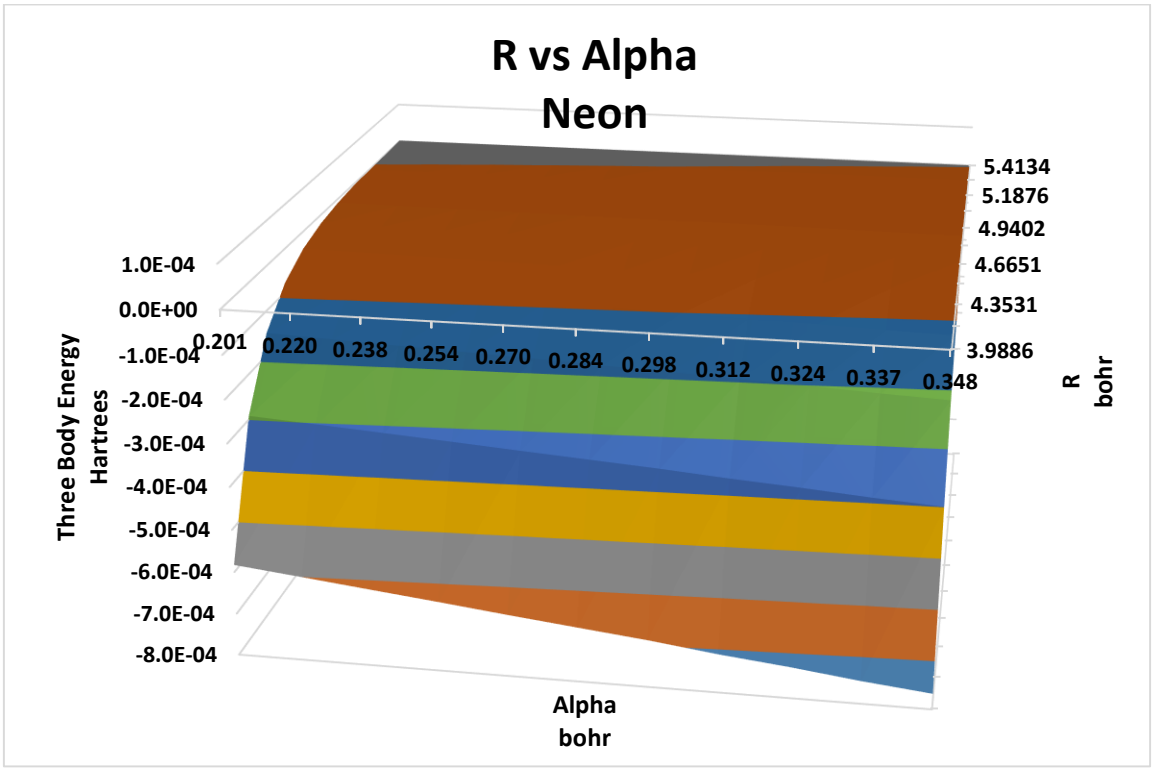

Figure 4.4: Surface graph of three body energy for R vs Alpha of a SET (Neon)

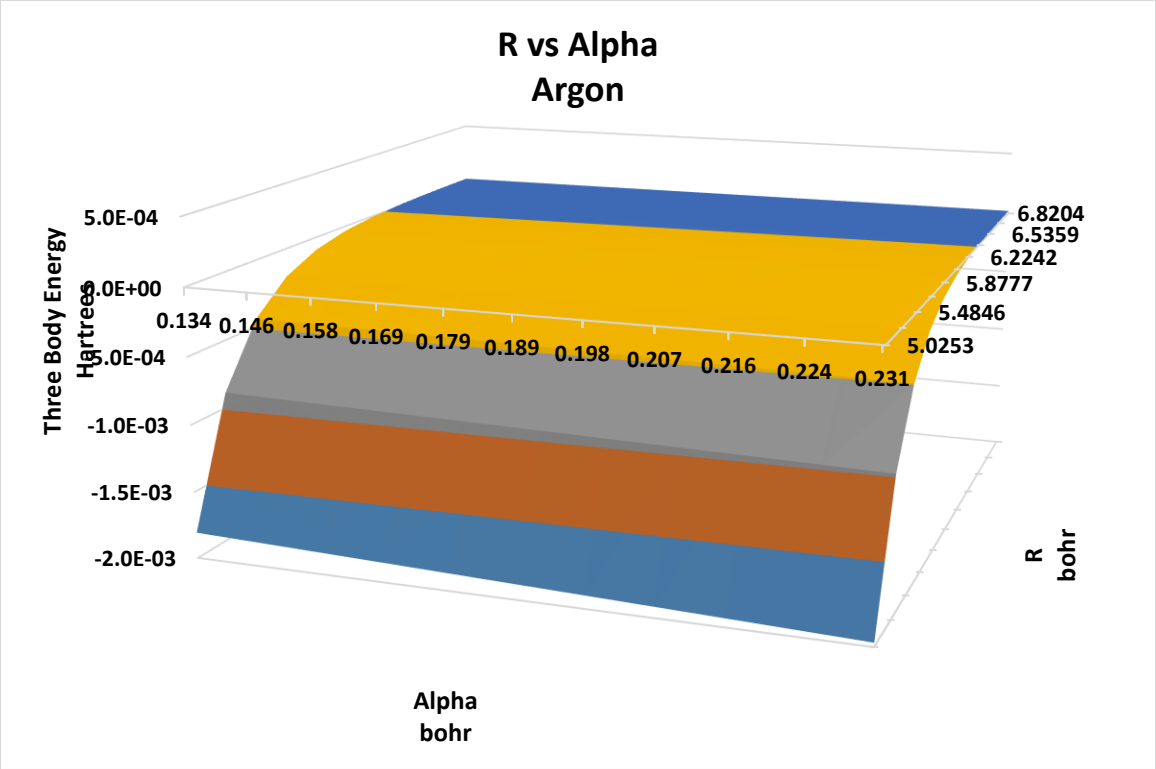

Figure 4.5: Surface graph of three body energy for R vs Alpha of a SET (Argon)

Table 4.6: Total three body contribution for Neon part 1

Three Body Potential Energy –Neon All Three-Body Energies in Hartrees R values (row) in bohr Alpha values (columns) in bohr						
	3.9886	**4.1788**	**4.3531**	**4.5145**	**4.6651**	**4.8066**
0.201	-1.51E-02	-7.32E-03	-3.56E-03	-1.73E-03	-8.08E-04	-3.28E-04
0.220	-1.56E-02	-7.60E-03	-3.72E-03	-1.82E-03	-8.64E-04	-3.63E-04
0.238	-1.61E-02	-7.88E-03	-3.88E-03	-1.92E-03	-9.23E-04	-3.98E-04
0.254	-1.66E-02	-8.17E-03	-4.05E-03	-2.02E-03	-9.83E-04	-4.35E-04
0.270	-1.71E-02	-8.46E-03	-4.22E-03	-2.12E-03	-1.04E-03	-4.73E-04
0.284	-1.76E-02	-8.76E-03	-4.39E-03	-2.22E-03	-1.11E-03	-5.12E-04
0.298	-1.82E-02	-9.06E-03	-4.57E-03	-2.33E-03	-1.17E-03	-5.52E-04
0.312	-1.87E-02	-9.36E-03	-4.75E-03	-2.44E-03	-1.24E-03	-5.93E-04
0.324	-1.92E-02	-9.67E-03	-4.93E-03	-2.55E-03	-1.31E-03	-6.36E-04
0.337	-1.97E-02	-9.98E-03	-5.11E-03	-2.66E-03	-1.38E-03	-6.79E-04
0.348	-2.02E-02	-1.03E-02	-5.30E-03	-2.78E-03	-1.45E-03	-7.24E-04

Table 4.7: Total three body contribution for Neon 2

Three Body Potential Energy –Neon All Three-Body Energies in Hartrees R values (row) in bohr Alpha values (columns) in bohr						
4.6651	**4.9402**	**5.0669**	**5.1876**	**5.3029**	**5.4134**	**5.5196**
0.201	-8.37E-05	3.98E-05	9.53E-05	1.16E-04	1.20E-04	1.16E-04
0.220	-1.05E-04	2.65E-05	8.70E-05	1.10E-04	1.17E-04	1.14E-04
0.238	-1.27E-04	1.27E-05	7.83E-05	1.05E-04	1.13E-04	1.12E-04
0.254	-1.50E-04	-1.61E-06	6.93E-05	9.92E-05	1.10E-04	1.10E-04
0.270	-1.74E-04	-1.64E-05	6.00E-05	9.32E-05	1.06E-04	1.07E-04
0.284	-1.98E-04	-3.18E-05	5.02E-05	8.70E-05	1.02E-04	1.05E-04
0.298	-2.23E-04	-4.78E-05	4.01E-05	8.05E-05	9.77E-05	1.02E-04
0.312	-2.49E-04	-6.43E-05	2.96E-05	7.37E-05	9.34E-05	9.93E-05
0.324	-2.76E-04	-8.13E-05	1.87E-05	6.66E-05	8.88E-05	9.63E-05
0.337	-3.04E-04	-9.90E-05	7.45E-06	5.93E-05	8.40E-05	9.32E-05
0.348	-3.32E-04	-1.17E-04	-4.23E-06	5.17E-05	7.90E-05	9.00E-05

Table 4.8: Total three body contribution for Neon 3

Three Body Potential Energy –Neon All Three-Body Energies in Hartrees R values (row) in bohr Alpha values (columns) in bohr						
	5.6218	**5.7000**	**5.8000**	**5.9000**	**5.9200**	**6.0000**
0.201	1.08E-04	1.01E-04	9.08E-05	8.05E-05	7.86E-05	7.11E-05
0.220	1.07E-04	1.00E-04	9.03E-05	8.03E-05	7.84E-05	7.10E-05
0.238	1.06E-04	9.94E-05	8.98E-05	8.01E-05	7.82E-05	7.09E-05
0.254	1.04E-04	9.84E-05	8.93E-05	7.98E-05	7.80E-05	7.08E-05
0.270	1.03E-04	9.74E-05	8.87E-05	7.95E-05	7.77E-05	7.07E-05
0.284	1.01E-04	9.63E-05	8.80E-05	7.91E-05	7.74E-05	7.05E-05
0.298	9.97E-05	9.51E-05	8.73E-05	7.87E-05	7.71E-05	7.03E-05
0.312	9.79E-05	9.39E-05	8.66E-05	7.83E-05	7.67E-05	7.01E-05
0.324	9.60E-05	9.26E-05	8.58E-05	7.79E-05	7.63E-05	6.99E-05
0.337	9.40E-05	9.11E-05	8.49E-05	7.73E-05	7.59E-05	6.96E-05
0.348	9.19E-05	8.97E-05	8.40E-05	7.68E-05	7.54E-05	6.93E-05

Table 4.9: Total three body contribution for Argon 1

Three Body Potential Energy –Argon All Three-Body Energies in Hartrees R values (row) in bohr Alpha values (columns) in bohr							
	5.025	**5.265**	**5.485**	**5.688**	**5.878**	**6.056**	**6.224**
0.134	-2.53E-02	-9.61E-03	-2.85E-03	-8.84E-05	1.05E-03	1.43E-03	1.52E-03
0.146	-2.57E-02	-9.82E-03	-2.96E-03	-1.49E-04	1.02E-03	1.41E-03	1.43E-03
0.158	-2.62E-02	-1.00E-02	-3.07E-03	-2.11E-04	9.83E-04	1.39E-03	1.41E-03
0.169	-2.66E-02	-1.02E-02	-3.19E-03	-2.73E-04	9.49E-04	1.36E-03	1.40E-03
0.179	-2.70E-02	-1.05E-02	-3.30E-03	-3.36E-04	9.14E-04	1.34E-03	1.39E-03
0.189	-2.75E-02	-1.07E-02	-3.42E-03	-4.00E-04	8.79E-04	1.32E-03	1.38E-03
0.198	-2.79E-02	-1.09E-02	-3.54E-03	-4.65E-04	8.43E-04	1.30E-03	1.37E-03
0.207	-2.84E-02	-1.11E-02	-3.65E-03	-5.30E-04	8.07E-04	1.28E-03	1.36E-03
0.216	-2.88E-02	-1.14E-02	-3.77E-03	-5.96E-04	7.70E-04	1.26E-03	1.35E-03
0.224	-2.93E-02	-1.16E-02	-3.90E-03	-6.62E-04	7.33E-04	1.24E-03	1.34E-03
0.231	-2.97E-02	-1.18E-02	-4.02E-03	-7.30E-04	6.95E-04	1.22E-03	1.33E-03

Table 4.10: Total three body contribution for Argon 2

Three Body Potential Energy –Argon All Three-Body Energies in Hartrees R values (row) in bohr Alpha values (columns) in bohr							
	6.384	**6.536**	**6.681**	**6.820**	**6.954**	**7.083**	**7.107**
0.134	1.28E-03	1.20E-03	9.31E-04	8.05E-04	6.83E-04	5.90E-04	5.74E-04
0.146	1.27E-03	1.19E-03	9.28E-04	8.04E-04	6.83E-04	5.90E-04	5.74E-04
0.158	1.27E-03	1.19E-03	9.26E-04	8.04E-04	6.83E-04	5.91E-04	5.74E-04
0.169	1.26E-03	1.19E-03	9.23E-04	8.03E-04	6.83E-04	5.91E-04	5.75E-04
0.179	1.26E-03	1.18E-03	9.21E-04	8.03E-04	6.83E-04	5.91E-04	5.75E-04
0.189	1.25E-03	1.18E-03	9.18E-04	8.02E-04	6.83E-04	5.92E-04	5.75E-04
0.198	1.24E-03	1.08E-03	9.16E-04	8.02E-04	6.83E-04	5.92E-04	5.76E-04
0.207	1.24E-03	1.08E-03	9.13E-04	8.01E-04	6.83E-04	5.92E-04	5.76E-04
0.216	1.23E-03	1.07E-03	9.10E-04	8.01E-04	6.83E-04	5.92E-04	5.76E-04
0.224	1.23E-03	1.07E-03	9.07E-04	8.00E-04	6.83E-04	5.93E-04	5.76E-04
0.231	1.22E-03	1.07E-03	9.04E-04	7.99E-04	6.83E-04	5.93E-04	5.77E-04

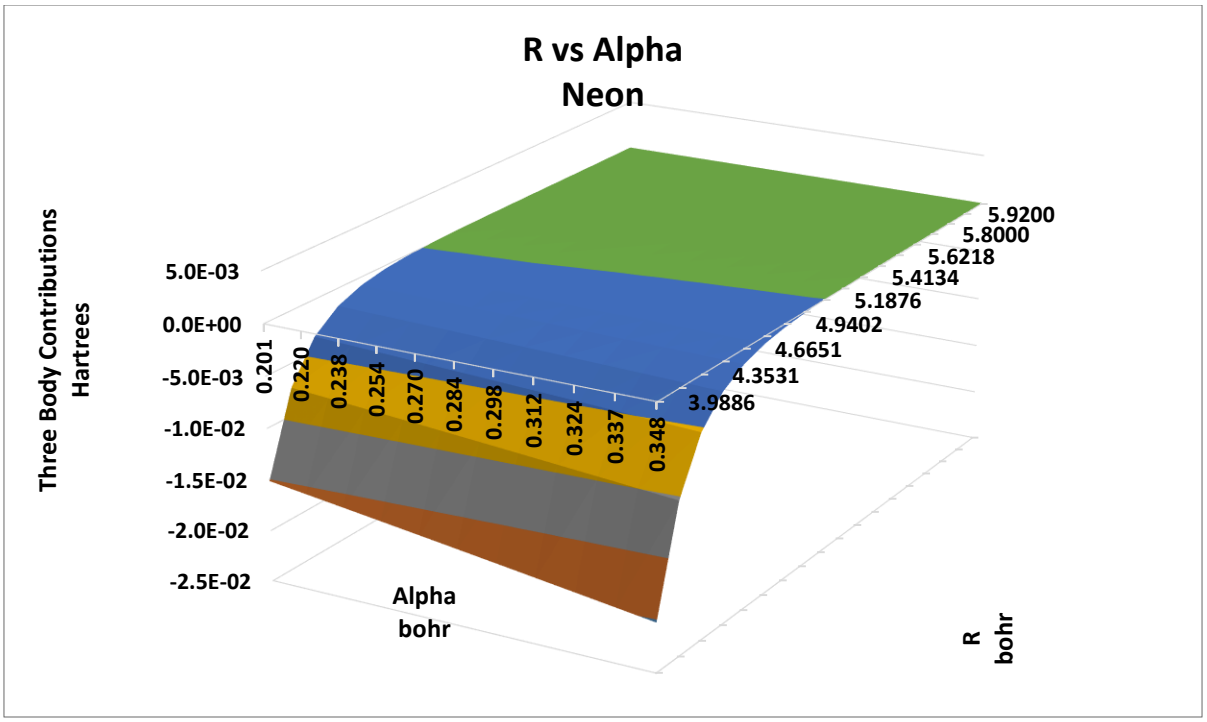

Figure 4.6: Surface graph of total three body contribution for R vs Alpha (Neon)

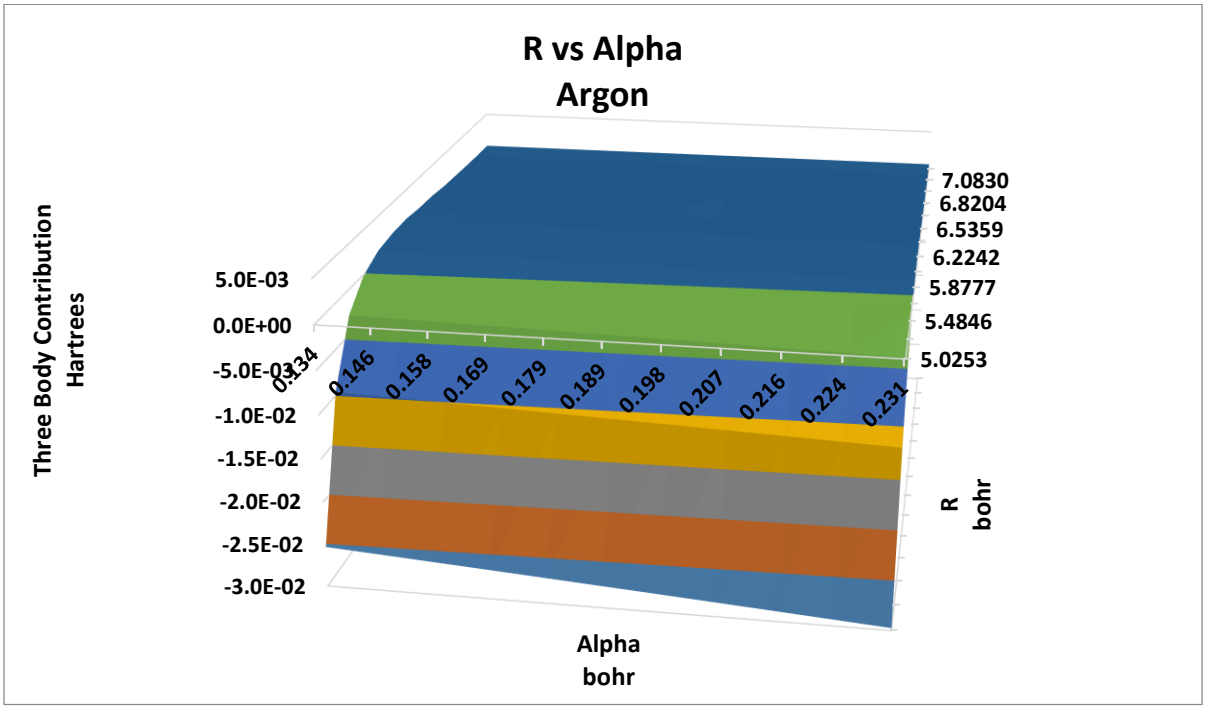

Figure 4.7: Surface graph of total three body contribution for R vs Alpha (Neon)

Table 4.11: Kinetic energy at different alpha values for Neon

Kinetic Energy -Neon All Energies in Hartrees						
Alpha bohr	**0.201**	**0.220**	**0.238**	**0.254**	**0.270**	**0.284**
	8.36E-05	6.96E-05	5.97E-05	5.22E-05	4.64E-05	4.18E-05
Alpha bohr	**0.298**	**0.312**	**0.324**	**0.337**	**0.348**	
	3.80E-05	3.48E-05	3.21E-05	2.99E-05	2.79E-05	

Table 4.12: Kinetic energy at different alpha values for Argon

Kinetic Energy -Argon All Energies in Hartrees						
Alpha bohr	**0.134**	**0.146**	**0.158**	**0.169**	**0.179**	**0.189**
	9.56E-05	7.97E-05	6.83E-05	5.98E-05	5.31E-05	4.78E-05
Alpha bohr	**0.198**	**0.207**	**0.216**	**0.224**	**0.231**	
	4.35E-05	3.99E-05	3.68E-05	3.42E-05	3.19E-05	

Table 4.13: Cohesive energy considering only two body contributions for Argon 1

Cohesive energy (Hartrees) only two body Contributions-Argon R values (row) in bohr Alpha values (columns) in bohr							
	5.0253	**5.2650**	**5.4846**	**5.6879**	**5.8777**	**6.0559**	**6.2242**
0.134	1.45E-01	7.32E-02	3.82E-02	2.01E-02	1.01E-02	4.48E-03	1.22E-03
0.146	1.49E-01	7.50E-02	3.92E-02	2.06E-02	1.04E-02	4.66E-03	1.33E-03
0.158	1.52E-01	7.69E-02	4.02E-02	2.12E-02	1.07E-02	4.86E-03	1.44E-03
0.169	1.56E-01	7.88E-02	4.12E-02	2.18E-02	1.11E-02	5.07E-03	1.57E-03
0.179	1.60E-01	8.08E-02	4.23E-02	2.24E-02	1.15E-02	5.29E-03	1.71E-03
0.189	1.65E-01	8.29E-02	4.34E-02	2.30E-02	1.18E-02	5.52E-03	1.86E-03
0.198	1.69E-01	8.50E-02	4.46E-02	2.37E-02	1.22E-02	5.77E-03	2.01E-03
0.207	1.73E-01	8.72E-02	4.58E-02	2.43E-02	1.26E-02	6.02E-03	2.17E-03
0.216	1.78E-01	8.94E-02	4.70E-02	2.50E-02	1.30E-02	6.27E-03	2.33E-03
0.224	1.83E-01	9.18E-02	4.82E-02	2.57E-02	1.35E-02	6.54E-03	2.50E-03
0.231	1.88E-01	9.42E-02	4.95E-02	2.65E-02	1.39E-02	6.81E-03	2.68E-03

Table 4.14: Cohesive energy considering only two body contributions for Argon 2

Cohesive energy (Hartrees) only two body Contributions-Argon R values (row) in bohr Alpha values (columns) in bohr							
	6.3839	**6.5359**	**6.6812**	**6.8204**	**6.9542**	**7.0830**	**7.1070**
0.134	-6.73E-04	-1.77E-03	-2.39E-03	-2.71E-03	-2.85E-03	-2.88E-03	-2.87E-03
0.146	-6.21E-04	-1.75E-03	-2.39E-03	-2.72E-03	-2.87E-03	-2.91E-03	-2.90E-03
0.158	-5.54E-04	-1.71E-03	-2.37E-03	-2.72E-03	-2.88E-03	-2.92E-03	-2.92E-03
0.169	-4.75E-04	-1.67E-03	-2.35E-03	-2.71E-03	-2.88E-03	-2.93E-03	-2.93E-03
0.179	-3.89E-04	-1.62E-03	-2.32E-03	-2.70E-03	-2.88E-03	-2.93E-03	-2.93E-03
0.189	-2.96E-04	-1.56E-03	-2.28E-03	-2.68E-03	-2.87E-03	-2.93E-03	-2.93E-03
0.198	-1.98E-04	-1.50E-03	-2.24E-03	-2.65E-03	-2.85E-03	-2.92E-03	-2.93E-03
0.207	-9.54E-05	-1.43E-03	-2.20E-03	-2.63E-03	-2.84E-03	-2.91E-03	-2.92E-03
0.216	1.15E-05	-1.36E-03	-2.15E-03	-2.60E-03	-2.82E-03	-2.90E-03	-2.91E-03
0.224	1.23E-04	-1.28E-03	-2.11E-03	-2.56E-03	-2.80E-03	-2.89E-03	-2.90E-03
0.231	2.38E-04	-1.21E-03	-2.05E-03	-2.53E-03	-2.78E-03	-2.88E-03	-2.88E-03

Table 4.15: Cohesive energy including three body contributions for Argon 1

Cohesive energy (Hartrees) including three body Contributions- Argon R values (row) in bohr Alpha values (columns) in bohr							
	5.0253	**5.2650**	**5.4846**	**5.6879**	**5.8777**	**6.0559**	**6.2242**
0.134	1.41E-01	7.16E-02	3.78E-02	2.00E-02	1.03E-02	4.72E-03	1.48E-03
0.146	1.44E-01	7.34E-02	3.87E-02	2.06E-02	1.06E-02	4.89E-03	1.56E-03
0.158	1.48E-01	7.52E-02	3.97E-02	2.11E-02	1.09E-02	5.09E-03	1.68E-03
0.169	1.52E-01	7.71E-02	4.07E-02	2.17E-02	1.13E-02	5.29E-03	1.81E-03
0.179	1.56E-01	7.90E-02	4.18E-02	2.23E-02	1.16E-02	5.51E-03	1.94E-03
0.189	1.60E-01	8.11E-02	4.29E-02	2.29E-02	1.20E-02	5.74E-03	2.09E-03
0.198	1.64E-01	8.32E-02	4.40E-02	2.36E-02	1.24E-02	5.98E-03	2.24E-03
0.207	1.69E-01	8.53E-02	4.51E-02	2.42E-02	1.28E-02	6.23E-03	2.40E-03
0.216	1.73E-01	8.76E-02	4.63E-02	2.49E-02	1.32E-02	6.48E-03	2.56E-03
0.224	1.78E-01	8.99E-02	4.76E-02	2.56E-02	1.36E-02	6.75E-03	2.73E-03
0.231	1.83E-01	9.22E-02	4.89E-02	2.64E-02	1.40E-02	7.02E-03	2.90E-03

Table 4.16: Cohesive energy including three body contributions for Argon 2

Cohesive energy (Hartrees) including three body Contributions- Argon R values (row) in bohr Alpha values (columns) in bohr							
	6.3839	**6.5359**	**6.6812**	**6.8204**	**6.9542**	**7.0830**	**7.1070**
0.134	-4.60E-04	-1.57E-03	-2.23E-03	-2.58E-03	-2.73E-03	-2.78E-03	-2.78E-03
0.146	-4.09E-04	-1.55E-03	-2.23E-03	-2.59E-03	-2.76E-03	-2.81E-03	-2.81E-03
0.158	-3.42E-04	-1.52E-03	-2.22E-03	-2.59E-03	-2.77E-03	-2.82E-03	-2.82E-03
0.169	-2.65E-04	-1.47E-03	-2.20E-03	-2.58E-03	-2.77E-03	-2.83E-03	-2.83E-03
0.179	-1.79E-04	-1.42E-03	-2.17E-03	-2.56E-03	-2.76E-03	-2.83E-03	-2.84E-03
0.189	-8.78E-05	-1.36E-03	-2.13E-03	-2.54E-03	-2.75E-03	-2.83E-03	-2.83E-03
0.198	9.27E-06	-1.32E-03	-2.09E-03	-2.52E-03	-2.74E-03	-2.82E-03	-2.83E-03
0.207	1.11E-04	-1.25E-03	-2.05E-03	-2.49E-03	-2.72E-03	-2.82E-03	-2.82E-03
0.216	2.17E-04	-1.18E-03	-2.00E-03	-2.46E-03	-2.70E-03	-2.80E-03	-2.81E-03
0.224	3.27E-04	-1.11E-03	-1.95E-03	-2.43E-03	-2.68E-03	-2.79E-03	-2.80E-03
0.231	4.41E-04	-1.03E-03	-1.90E-03	-2.40E-03	-2.66E-03	-2.78E-03	-2.79E-03

Table 4.17: Cohesive energy considering only two body contributions for Neon 1

Cohesive energy (Hartrees) only two body Contributions-Neon R values (row) in bohr Alpha values (columns) in bohr						
	3.9886	**4.1788**	**4.3531**	**4.5145**	**4.6651**	**4.8066**
0.201	5.26E-02	3.12E-02	1.89E-02	1.16E-02	7.05E-03	4.21E-03
0.220	5.49E-02	3.27E-02	1.99E-02	1.22E-02	7.49E-03	4.51E-03
0.238	5.73E-02	3.42E-02	2.09E-02	1.29E-02	7.96E-03	4.83E-03
0.254	5.98E-02	3.58E-02	2.20E-02	1.36E-02	8.45E-03	5.18E-03
0.270	6.24E-02	3.75E-02	2.31E-02	1.44E-02	8.98E-03	5.55E-03
0.284	6.51E-02	3.92E-02	2.42E-02	1.52E-02	9.53E-03	5.94E-03
0.298	6.78E-02	4.10E-02	2.54E-02	1.60E-02	1.01E-02	6.34E-03
0.312	7.07E-02	4.29E-02	2.67E-02	1.69E-02	1.07E-02	6.77E-03
0.324	7.37E-02	4.49E-02	2.80E-02	1.77E-02	1.13E-02	7.22E-03
0.337	7.68E-02	4.69E-02	2.93E-02	1.87E-02	1.20E-02	7.68E-03
0.348	7.99E-02	4.89E-02	3.07E-02	1.96E-02	1.27E-02	8.17E-03

Table 4.18: Cohesive energy considering only two body contributions for Neon 2

Cohesive energy (Hartrees) only two body Contributions-Neon R values (row) in bohr Alpha values (columns) in bohr						
	4.9402	**5.0669**	**5.1876**	**5.3029**	**5.4134**	**5.5196**
0.201	2.40E-03	1.24E-03	4.82E-04	-2.95E-06	-3.11E-04	-5.02E-04
0.220	2.60E-03	1.37E-03	5.71E-04	5.21E-05	-2.81E-04	-4.89E-04
0.238	2.83E-03	1.53E-03	6.78E-04	1.24E-04	-2.34E-04	-4.62E-04
0.254	3.07E-03	1.70E-03	7.99E-04	2.08E-04	-1.77E-04	-4.24E-04
0.270	3.33E-03	1.89E-03	9.32E-04	3.03E-04	-1.10E-04	-3.78E-04
0.284	3.61E-03	2.08E-03	1.08E-03	4.06E-04	-3.56E-05	-3.25E-04
0.298	3.90E-03	2.30E-03	1.23E-03	5.18E-04	4.58E-05	-2.66E-04
0.312	4.21E-03	2.52E-03	1.39E-03	6.38E-04	1.34E-04	-2.01E-04
0.324	4.53E-03	2.75E-03	1.56E-03	7.65E-04	2.28E-04	-1.32E-04
0.337	4.87E-03	3.00E-03	1.75E-03	9.00E-04	3.29E-04	-5.65E-05
0.348	5.22E-03	3.26E-03	1.94E-03	1.04E-03	4.35E-04	2.36E-05

Table 4.19: Cohesive energy considering only two body contributions for Neon 3

Cohesive energy (Hartrees) only two body Contributions-Neon R values (row) in bohr Alpha values (columns) in bohr						
	5.6218	**5.7000**	**5.8000**	**5.9000**	**5.9200**	**6.0000**
0.201	-6.13E-04	-6.63E-04	-6.93E-04	-6.96E-04	-6.94E-04	-6.79E-04
0.220	-6.14E-04	-6.72E-04	-7.11E-04	-7.20E-04	-7.19E-04	-7.07E-04
0.238	-6.01E-04	-6.67E-04	-7.14E-04	-7.30E-04	-7.30E-04	-7.23E-04
0.254	-5.77E-04	-6.53E-04	-7.09E-04	-7.32E-04	-7.33E-04	-7.30E-04
0.270	-5.47E-04	-6.31E-04	-6.97E-04	-7.27E-04	-7.30E-04	-7.31E-04
0.284	-5.10E-04	-6.04E-04	-6.80E-04	-7.18E-04	-7.22E-04	-7.28E-04
0.298	-4.67E-04	-5.72E-04	-6.59E-04	-7.05E-04	-7.10E-04	-7.21E-04
0.312	-4.20E-04	-5.36E-04	-6.33E-04	-6.88E-04	-6.95E-04	-7.11E-04
0.324	-3.69E-04	-4.96E-04	-6.05E-04	-6.68E-04	-6.76E-04	-6.98E-04
0.337	-3.13E-04	-4.52E-04	-5.73E-04	-6.45E-04	-6.55E-04	-6.83E-04
0.348	-2.53E-04	-4.04E-04	-5.37E-04	-6.20E-04	-6.32E-04	-6.65E-04

Table 4.20: Cohesive energy including three body contributions for Neon 1

Cohesive energy (Hartrees) including three body Contributions-Neon R values (row) in bohr Alpha values (columns) in bohr						
	3.989	**4.179**	**4.353**	**4.515**	**4.665**	**4.807**
0.201	5.00E-02	3.00E-02	1.83E-02	1.13E-02	6.92E-03	4.16E-03
0.220	5.23E-02	3.14E-02	1.93E-02	1.19E-02	7.34E-03	4.45E-03
0.238	5.46E-02	3.29E-02	2.03E-02	1.26E-02	7.80E-03	4.77E-03
0.254	5.70E-02	3.45E-02	2.13E-02	1.33E-02	8.29E-03	5.11E-03
0.270	5.95E-02	3.61E-02	2.24E-02	1.40E-02	8.80E-03	5.47E-03
0.284	6.21E-02	3.78E-02	2.35E-02	1.48E-02	9.34E-03	5.85E-03
0.298	6.48E-02	3.95E-02	2.47E-02	1.56E-02	9.91E-03	6.25E-03
0.312	6.76E-02	4.14E-02	2.59E-02	1.64E-02	1.05E-02	6.67E-03
0.324	7.05E-02	4.32E-02	2.72E-02	1.73E-02	1.11E-02	7.11E-03
0.337	7.35E-02	4.52E-02	2.85E-02	1.82E-02	1.18E-02	7.57E-03
0.348	7.66E-02	4.72E-02	2.99E-02	1.92E-02	1.24E-02	8.05E-03

Table 4.21: Cohesive energy including three body contributions for Neon 2

Cohesive energy (Hartrees) including three body Contributions-Neon R values (row) in bohr Alpha values (columns) in bohr						
	4.940	**5.067**	**5.188**	**5.303**	**5.413**	**5.520**
0.201	2.39E-03	1.24E-03	4.98E-04	1.63E-05	-2.91E-04	-4.82E-04
0.220	2.59E-03	1.38E-03	5.85E-04	7.05E-05	-2.61E-04	-4.70E-04
0.238	2.81E-03	1.53E-03	6.91E-04	1.41E-04	-2.15E-04	-4.43E-04
0.254	3.05E-03	1.70E-03	8.11E-04	2.25E-04	-1.58E-04	-4.06E-04
0.270	3.30E-03	1.88E-03	9.42E-04	3.18E-04	-9.24E-05	-3.60E-04
0.284	3.58E-03	2.08E-03	1.08E-03	4.21E-04	-1.86E-05	-3.07E-04
0.298	3.86E-03	2.29E-03	1.24E-03	5.32E-04	6.21E-05	-2.49E-04
0.312	4.17E-03	2.51E-03	1.40E-03	6.50E-04	1.49E-04	-1.85E-04
0.324	4.48E-03	2.74E-03	1.57E-03	7.76E-04	2.43E-04	-1.15E-04
0.337	4.82E-03	2.98E-03	1.75E-03	9.10E-04	3.43E-04	-4.09E-05
0.348	5.17E-03	3.24E-03	1.94E-03	1.05E-03	4.48E-04	3.86E-05

Table 4.22: Cohesive energy including three body contributions for Neon 3

Cohesive energy (Hartrees) including three body Contributions-Neon R values (row) in bohr Alpha values (columns) in bohr						
	5.622	**5.700**	**5.800**	**5.900**	**5.920**	**6.000**
0.201	-5.95E-04	-6.46E-04	-6.78E-04	-6.82E-04	-6.81E-04	-6.67E-04
0.220	-5.96E-04	-6.55E-04	-6.96E-04	-7.06E-04	-7.05E-04	-6.95E-04
0.238	-5.83E-04	-6.51E-04	-6.99E-04	-7.17E-04	-7.17E-04	-7.11E-04
0.254	-5.60E-04	-6.36E-04	-6.94E-04	-7.18E-04	-7.20E-04	-7.18E-04
0.270	-5.30E-04	-6.15E-04	-6.83E-04	-7.14E-04	-7.17E-04	-7.20E-04
0.284	-4.93E-04	-5.88E-04	-6.65E-04	-7.05E-04	-7.09E-04	-7.16E-04
0.298	-4.51E-04	-5.56E-04	-6.44E-04	-6.91E-04	-6.97E-04	-7.09E-04
0.312	-4.04E-04	-5.20E-04	-6.19E-04	-6.75E-04	-6.82E-04	-6.99E-04
0.324	-3.53E-04	-4.80E-04	-5.90E-04	-6.55E-04	-6.64E-04	-6.86E-04
0.337	-2.97E-04	-4.36E-04	-5.58E-04	-6.33E-04	-6.43E-04	-6.71E-04
0.348	-2.38E-04	-3.89E-04	-5.23E-04	-6.07E-04	-6.19E-04	-6.53E-04

Table 4.23: Einstein Model vs. Experimental

Atom	**Two Body Potential**	**Three Body Potential**	**Experimental (Hartrees/Atom)**	**Einstein Model (Hartrees/Atom)**	**Percent Error**
Neon	**Korona/TT**	**EAT**	-7.163E-04	-6.819E-04	4.80%
Neon	**LJ**	**EAT**	-7.163E-04	-6.562E-04	8.39%
Neon	**AGY**	**EAT**	-7.163E-04	-6.550E-04	8.56%
Neon	**Korona/TT**	**TBE**	-7.163E-04	-7.136E-04	0.37%
Neon	**Korona/TT**	**MAT**	-7.163E-04	-6.852E-04	4.33%
Neon	**MTT**	**TBE**	-7.163E-04	-7.083E-04	1.11%
Neon	**MTT**	**EAT**	-7.163E-04	-6.766E-04	5.54%
Neon	**AGY**	**MAT**	-7.163E-04	-6.583E-04	8.09%
Neon	**AGY**	**TBE**	-7.163E-04	-6.872E-04	4.13%
Neon	**LJ**	**TBE**	-7.163E-04	-6.879E-04	3.96%
Argon	**LJ**	**Cencek**	-2.938E-03	-2.959E-03	0.71%
Argon	**Aziz-Slaman**	**Cencek**	-2.938E-03	-3.499E-03	19.08%
Argon	**AGY**	**Cencek**	-2.938E-03	-2.960E-03	0.73%
Argon	**GSM**	**Cencek**	-2.938E-03	-3.272E-03	11.36%

Table 4.24: TPA vs MCI for high and low alpha values (Symmetry-Neon)

Symmetry Coordinates– Neon EAT Three Body Potential- Hartrees				
R bohr	**Alpha bohr**	**MCI**	**TPA**	**Percent Error**
3.9886	0.201	-5.827E-04	-5.940E-04	1.91%
3.9886	0.348	-7.634E-04	-8.320E-04	9.02%
4.8066	0.201	-2.031E-05	-2.150E-05	5.69%
4.8066	0.348	-3.559E-05	-4.390E-05	23.45%
5.4134	0.201	1.920E-06	1.849E-06	3.71%
5.4134	0.348	2.300E-07	-6.520E-07	383.41%

Table 4.25: TPA vs MCI for high and low alpha values (Cartesian-Neon)

Cartesian Coordinates – Neon EAT Three Body Potential- Hartrees				
R bohr	**Alpha bohr**	**MCI**	**TPA**	**Percent Error**
3.9886	0.201	-5.827E-04	-5.790E-04	0.66%
3.9886	0.348	-7.634E-04	-7.442E-04	2.52%
5.4134	0.201	1.920E-06	1.980E-06	3.13%
5.4134	0.348	2.300E-07	6.981E-07	203.46%

Table 4.26: TPA vs MCI for Glyde's alpha values (Symmetry-Neon)

Symmetry Coordinates – Neon EAT Three Body Potential- Hartrees Glyde's $\langle U^2 \rangle$				
R bohr	**Alpha bohr**	**MCI**	**TPA**	**Percent Error**
3.9886	0.210	-5.906E-04	-6.034E-04	2.16%
4.8066	0.253	-2.443E-05	-2.680E-05	9.64%
5.4134	0.284	1.168E-06	8.278E-07	29.13%

Table 4.27: TPA vs MCI for Glyde's alpha values (Cartesian-Neon)

Cartesian Coordinates – Neon EAT Three Body Potential- Hartrees Glyde's $<U^2>$				
R bohr	**Alpha bohr**	**MCI**	**TPA**	**Percent Error**
3.9886	0.210	-5.906E-04	-5.862E-04	0.72%
5.4134	0.284	1.168E-06	1.355E-06	16.00%

Table 4.28: TPA vs MCI at the energy minimum (Symmetry-Neon)

Energy Minimum – Neon EAT Three Body Potential- Hartrees Glyde's $<U^2>$					
TPA Method	**R bohr**	**Alpha bohr**	**MCI**	**TPA**	**Percent Error**
Symmetry	5.92	0.312	1.801E-06	1.819E-06	1.01%
Cartesian	5.92	0.312	1.801E-06	1.861E-06	3.32%

Table 4.29: TPA vs MCI for high and low alpha values (Symmetry-Argon)

Symmetry Coordinates -Argon Cencek Three Body Potential- Hartrees				
R bohr	**Alpha bohr**	**MCI**	**TPA**	**Percent**
5.0253	0.134	-1.808E-03	-1.800E-03	0.40%
5.0253	0.231	-1.970E-03	-2.010E-03	2.03%
6.0559	0.134	-3.875E-05	-3.730E-05	3.70%
6.0559	0.231	-5.025E-05	-4.750E-05	5.37%
6.8204	0.134	1.785E-05	1.828E-05	2.46%
6.8204	0.231	1.687E-05	1.795E-05	6.39%

Table 4.30: TPA vs MCI for high and low alpha values (Cartesian-Argon)

Cartesian-Argon Cencek Three Body Potential- Hartrees				
R	**alpha**	**MCI**	**TPA**	**Percent**
5.0253	0.134	-1.808E-03	-1.842E-03	1.88%
5.0253	0.231	-1.970E-03	-2.071E-03	5.12%
5.2650	0.134	-8.879E-04	-9.190E-04	3.49%
5.2650	0.231	-9.789E-04	-1.060E-03	7.89%
5.4846	0.134	-4.401E-04	-4.240E-04	3.63%
5.4846	0.231	-4.929E-04	-4.520E-04	8.36%
6.0559	0.134	-3.875E-05	-3.830E-05	1.17%
6.0559	0.231	-5.025E-05	-4.840E-05	3.73%
6.8204	0.134	1.785E-05	1.785E-05	0.05%
6.8204	0.231	1.687E-05	1.701E-05	0.83%

Table 4.31: TPA vs MCI for Glyde's alpha values (Symmetry-Argon)

Symmetry Coordinates – Argon Cencek Three Body Potential- Hartrees Glyde's $<U^2>$				
R bohr	**Alpha bohr**	**MCI**	**TPA**	**Percent Error**
5.0253	0.139	-1.814E-03	-1.808E-03	0.33%
6.0559	0.168	-4.198E-05	-3.991E-05	4.85%
6.8204	0.189	1.738E-05	1.818E-05	4.60%

Table 4.32: TPA vs MCI for Glyde's alpha values (Cartesian-Argon)

Cartesian Coordinates – Argon Cencek Three Body Potential- Hartrees Glyde's $<U^2>$				
R bohr	**Alpha bohr**	**MCI**	**TPA**	**Percent Error**
5.0253	0.139	-1.814E-03	-1.854E-03	2.00%
6.0559	0.168	-4.198E-05	-4.123E-05	1.92%
6.8204	0.189	1.738E-05	1.744E-05	0.33%

Table 4.33: TPA vs MCI at energy minimum (Symmetry-Argon)

Energy Minimum – Argon Cencek Three Body Potential- Hartrees Glydes <U^2>					
TPA Method	**R bohr**	**Alpha bohr**	**MCI**	**TPA**	**Percent Error**
Symmetry	7.10	0.198	1.551E-05	1.600E-05	3.17%
Cartesian	7.10	0.198	1.551E-05	1.554E-05	0.19%

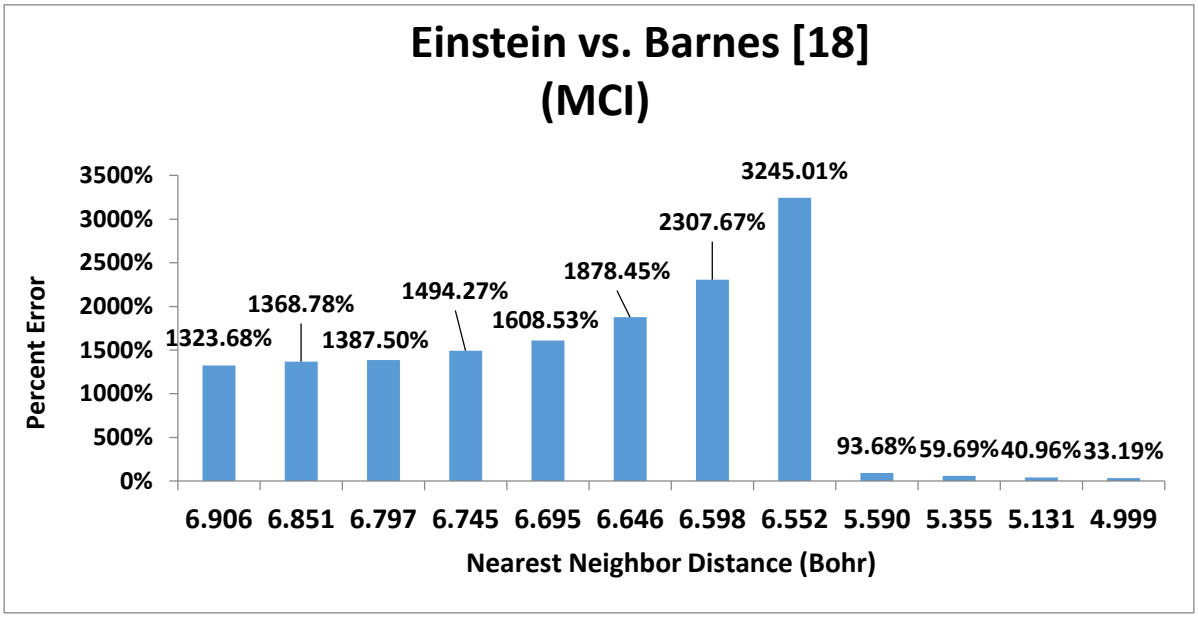

Figure 4.8: Einstein vs Barnes

Table 4.34: TPA vs MCI for different polynomial orders at a low density (Helium)

Helium (R=6.906 bohr) Average three body energy in Hartrees.					
Polynomial Order	**MCI**	**TPA Q**	**%Difference**	**TPA Cartesian**	**%Difference**
6th	-3.445E-07	-7.033E-05	20313.90%	7.707E-08	122.37%
8th	-3.445E-07	4.418E-06	1382.29%	5.303E-08	115.39%
10th	-3.445E-07	-3.381E-07	1.88%	5.298E-08	115.38%
12th	-3.445E-07	-2.482E-06	620.36%	5.361E-08	115.56%
14th	-3.445E-07	-2.374E-06	588.98%	5.349E-08	115.52%
16th	-3.445E-07	-2.302E-06	568.19%	5.350E-08	115.53%

Table 4.35: TPA vs MCI for different polynomial orders at a high density (Helium)

Helium (R=4.999 bohr) Average three body energy in Hartrees.					
Polynomial Order	**MCI**	**TPA Q**	**%Difference**	**TPA Cartesian**	**%Difference**
6th	-9.698E-06	-1.544E-05	59.20%	-1.328E-05	36.96%
8th	-9.698E-06	-1.219E-05	25.69%	-6.634E-06	31.59%
10th	-9.698E-06	-1.234E-05	27.20%	-8.867E-06	8.56%
12th	-9.698E-06	-1.246E-05	28.53%	-8.919E-06	8.03%
14th	-9.698E-06	-1.233E-05	27.13%	-8.917E-06	8.05%
16th	-9.698E-06	-1.238E-05	27.65%	-8.877E-06	8.47%

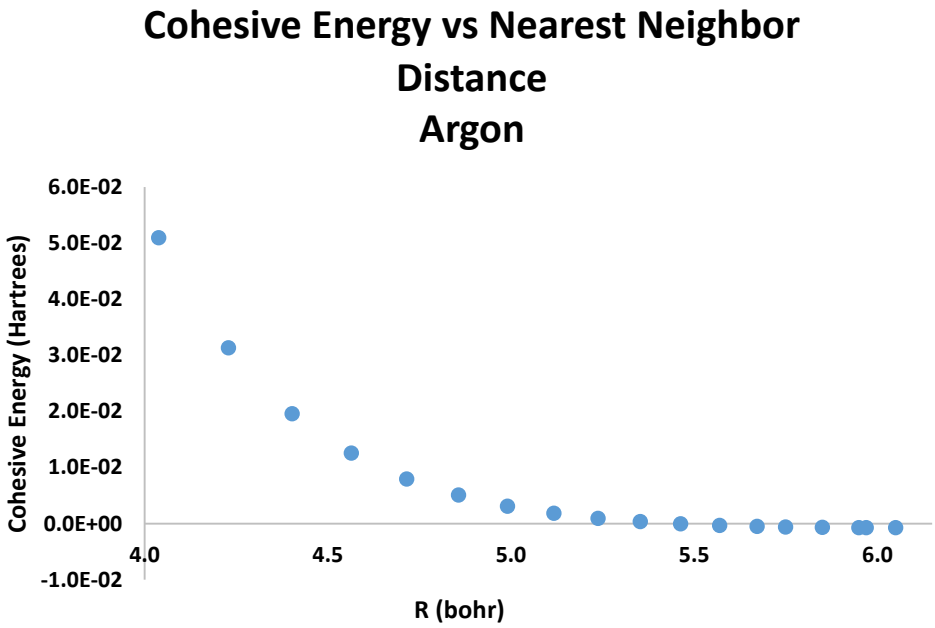

Figure 4.9: Cohesive Energy Vs Nearest Neighbor Distance

NEON

Send_all.py

'''
This program sends calculations for the total two body contributions in a crystal at different R values. It checks the two files for R values and C or alpha values. Then for every R value it will calculate a three body energy for every C or alpha value. Things to watch out for: watch out for blank lines in the R.dat or C.dat, sometimes python will include blank lines into the array. Also make sure you are using the correct alpha value this program takes the gaussian in the form of exp(-(alpha(x^2))).
'''

```
import os
import time
R_file=open('R.dat','r').readlines()#R values
C_file=open('C.dat','r').readlines()#alpha values

R_values=[]# R array
C_values=[]# C or alpha array

#append R values and alpha values
for line in R_file:
   line=line.rstrip()
   R_values.append(line)

for line in C_file:
   line=line.rstrip()
   C_values.append(line)

#loops through each R value with the alpha values looped for each
#R values
for i in R_values:
   for j in C_values:
      R_C=open('R_C.dat','w')#opens a new file
      R_C.write(str(i)+'\n'+str(j))#writes R value and alpha
      R_C.close()#close file
      os.system('./RUN_RPG')#runs a program that takes the R
 #and alpha values from the R_C.dat file.
      time.sleep(1)
```

RUN_RPG script

```
g++ XYZ_coordinates.cpp
./a.out > distances.txt
sleep 1

python no_duplicates.py

sleep 1

g++ Two_body_contribution.cpp
./a.out
```

XYZ_coordinates.cpp

```
#include<iostream>
#include<math.h>
#include<fstream>
//header files for A,B,C layers
#include "A_Layer.h"
#include "B_Layer.h"
#include "C_Layer.h"
using namespace std;



int main()
{
//A,B,C layer objects
  A_Lay A;
  B_Lay B;
  C_Lay C;
  ifstream fp;
  fp.open("R_C.dat");//file for the R value
  double R;
  int rotations=4, Layers=4;//this can be adjusted depending how much you want the crystal to
expand
  fp>>R;//reads in the R

//set each layer and have them printed out
  A.set_function(R);
  A.get_function(rotations,R,Layers);
  B.set_function(R);
  B.get_function(rotations,R,Layers);
  C.set_function(R);
  C.get_function(rotations,R,Layers);


  return 0;
}
```

No_duplicates.py

```
fp=open("distances.txt",'r')
lines=fp.readlines()#reads lines from distances
distance=[]
XYZ=[]
for line in lines:
    parts=line.split()#splits lines
    distance.append(parts[0])# grabs distances
    XYZ.append(parts[1])#X
    XYZ.append(parts[2])#Y
    XYZ.append(parts[3])#Z
parallel_singles=[]
singles=[]
x=0
y=1
z=2
for i in distance:
    if i not in singles:
        singles.append(i)#no duplicates
              #XYZ coordinates that go with that distance
        parallel_singles.append(XYZ[x])
        parallel_singles.append(XYZ[y])
        parallel_singles.append(XYZ[z])
    x+=3
    y+=3
    z+=3

x=0
y=1
z=2
fileoutput=open("Rcoordinates.txt",'w')#new file
for j in singles:#goes through the single distances
    j.rstrip()# I'm not if this is needed, but it strips the white space
    count=0#counter
    for k in distance:
        k.rstrip()
        if j==k:#checks the frequency of this base distance
            count+=1
    fileoutput.write(str(parallel_singles[x])+" "+str(parallel_singles[y])+"
"+str(parallel_singles[z])+" "+str(count)+"\n")# print out
    x+=3
    y+=3
    z+=3

fp.close()
fileoutput.close()
```

Two_body_contribution.cpp

```
#include<iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include "Random_Points.h"//random point header file
using namespace std;
//Function Prototypes for two body functions
double MTTY(double &);
double CT(double &);//this is really the AGY
double S2n(int &,double &,double &);
double TTY(double &);//AKA korona/TTY
double fact(int);
double LJ(double &);
double factorial(int &);

int main()
{
  ifstream fp,fp1;
  fp.open("Rcoordinates.txt");//opens file
  fp1.open("R_C.dat");//R value and C value
  unsigned ran_state[4];
  RPG Rando;//random points
  double R,xbase[200],ybase[200],zbase[200],freq[200],two_body,C,energy,R1;
  double xapoint,yapoint, zapoint,xbpoint,ybpoint,zbpoint,xa,ya,za,xb,yb,zb,Total_energy=0;
  int count=0,n=1E6,end,i;
  fp1>>R1>>C;
  double alpha=sqrt(1/(2*C));//converts C to Alpha
  while(true)
    {
//reads in X,Y,Z and the Frequency of occurence
      fp>>xbase[count]>>ybase[count]>>zbase[count]>>freq[count];
           if (fp.eof())
             {break;}
      count++;

    }
  end=count;//counts lines in the file

  for(int j=0;j<end;j++)//loops through each line
    {
      two_body=0;
  for (i=0; i<n; i++)//1 million random points
    {
//random points for XYZ for both atoms; a Number between 0 and 1
```

```
      xapoint = Rando.rangauss(ran_state);
      yapoint = Rando.rangauss(ran_state);
      zapoint = Rando.rangauss(ran_state);
      xbpoint = Rando.rangauss(ran_state);
      ybpoint = Rando.rangauss(ran_state);
      zbpoint = Rando.rangauss(ran_state);

//include alpha and base positons
      xa=alpha*xapoint+xbase[j];
      ya=alpha*yapoint+ybase[j];
      za=alpha*zapoint+zbase[j];
      xb=alpha*xbpoint;
      yb=alpha*ybpoint;
      zb=alpha*zbpoint;
//calc distance
      R=sqrt(pow((xb-xa),2)+pow((yb-ya),2)+pow((zb-za),2) );
      energy=MTTY(R);//goes to whatever function you like
      two_body+=energy;
    }
  Total_energy+=(two_body/n)*freq[j];//sums up average two body energy

     }

  cout<<Total_energy<<" "<<R1<<" "<<alpha<<endl;//print out

    fp.close();
    fp1.close();
  return(0);

          }

//I would read the papers for each one of these functions

//TTY function
double TTY(double &R)
{
//parameter
  double two_body,A=78.52,alpha=2.13371,beta=-.035;
  double b=1.88,c[20],tot_attractive=0;
  double repulsive, dispersion,attractive1=0,attractive2,kfact;
  c[6]=6.96;c[8]=49.87;c[10]=2393.96;
  c[12]=33172.16229,c[14]=514081.6678,c[16]=7474943.603;

//repulsive term
  repulsive=A*exp(-alpha*R+beta*pow(R,2));
  //  cout<<repulsive<<endl;
```

```
//attractive terms
  for(int i=3;i<9;i++)
    {
      attractive1=c[2*i]/double(pow(R,(2*i)));
      attractive2=0;
      for(int k=0;k<=(2*i);k++)
        {
          kfact=fact(k);//calls factorial function
          attractive2+=double(pow((b*R),k)/kfact);
        }
      tot_attractive+= attractive1*(1.0-exp(-b*R)*attractive2);
    }

  two_body=repulsive-tot_attractive;//repusive - attractive

  return two_body;//returns two body energy
}


double fact(int k)//factorial function
{
  double kfact=1.0;
  if (k==0)
    {return kfact;}

  for(int i=1;i<(k+1);i++)
    {
      kfact=kfact*double(i);
    }

  return kfact; //returns factorial

}
//Lennard Jones
double LJ(double &R)
{
//I've found 3 different parameters for Lennard Jones
//comment out the two parameter you are not using
  //  double sigma=5.24574669,epsilon=1.166370E-04;//LJ 1
  // double sigma=5.25897921,epsilon=1.132150E-04;//LJ 2
    double sigma=5.25897921,epsilon=1.179699E-04;//LJ 3


  double VLJ;

  VLJ=4.0*epsilon*(pow((sigma/R),12)-pow((sigma/R),6));
```

```
  return VLJ;//returns two body

}

//AGY
double CT(double &R)
{//parameters
  double A=88.5513,alpha=2.20626,beta=-0.0249851,b=1.85166;
  double G[10][10],C[20];
  double CT,VHF1,VHF2=0,VHF,VC,VC1=0,VC2=0,S2;
  G[1][1]=-3.205E-04;
  G[2][1]=0.82172;
  G[3][1]=3.2949;
  G[1][2]=-2.771E-03;
  G[2][2]=1.1097;
  G[3][2]=2.2072;
  C[6]=6.28174;
  C[8]=90.0503;
  C[10]=1679.45;
  C[12]=4.190E+04;
  C[14]=1.363E+06;
  C[16]=5.629E+07;


//Repulsive
  VHF1=A*exp((-alpha*R+pow(beta,1)*pow(R,2)));

  for(int i=1;i<3;i++)
    {
      VHF2+=G[1][i]*exp(-G[2][i]*pow((R-G[3][i]),2));
    }

  VHF=VHF1+VHF2;//total repulsive
  for (int i=1;i<3;i++)
    {
      VC1+=G[1][i]*exp(-G[2][i]*pow((R-G[3][i]),2));
    }

  for (int i=3;i<9;i++)
    {
      S2=S2n(i,R,b);//s2n function
      VC2+=S2*(C[2*i]/pow(R,(2*i)));
    }
  VC=-(VC1+VC2);//attractive
  CT=VHF+VC;//two body energy
```

```
  return CT;//return two body
}


double S2n(int &i,double &R,double &b)
{
  double S2=0,kfact;

  for(int k=0;k<=2*i;k++)
    {
      kfact=factorial(k);//factorial function
      S2+=pow((b*R),k)/kfact;
    }

  S2=1.0-exp(-b*R)*S2;

  return S2;
}


//factorial function
double factorial(int &k)
{double fact=1.0;
  if (k==0)
    {fact=1;return fact;}
  for (int i=1;i<=k;i++)
    {
      fact*=i;
    }

  return fact;
}

double MTTY(double &r) //modified TTY potential
{
//parameters
  double A=4.0291505E7,b=4.9248731676E1,Rm=.3089456,R;
  double C[20],a1=-4.28654039586E1,a2=-3.33818674327,a_1=-5.34644860719E-
02,a_2=5.01774999419E-03;
  double epsilon=42.152521,sigma,VC=0,VHF,S2;
  double Modified_TTY;
  R=r*.0529;
  C[6]=4.40676750157E-02;
  C[8]=1.64892507701E-03;
  C[10]=7.90473640524E-05;
  C[12]=4.85489170103E-06;
```

```
  C[14]=3.82012334054E-07;
  C[16]=3.85106552963E-08;

//repulsive
  VHF=A*exp(a1*R+a2*pow(R,2)+a_1*pow(R,(-1))+a_2*pow(R,(-2)));


  for (int i=3;i<9;i++)
    {
      S2=S2n(i,R,b);
      VC+=S2*(C[2*i]/pow(R,(2*i)));
    }
//attractive
  VC=-VC;
  Modified_TTY=VHF+VC;

  return Modified_TTY*3.166815E-6;//*epsilon (converts Kelvin to Hartrees)
}
```

Send_all.py

```
'''
This program sends calculations for a single small equilateral triangle.  It checks the two files for
R values and C or alpha values.  Then for every R value it will calculate a three body energy for
every C or alpha value.  Things to watch out for: watch out for blank lines in the R.dat or C.dat,
sometimes python will include blank lines into the array.  Also make sure you are using the
correct alpha value this program takes the gaussian in the form of exp(-(x^2) /(2*(alpha^2))).
'''
import os
import time
R_file=open('R.dat','r').readlines()#R values
C_file=open('C.dat','r').readlines()#alpha values

R_values=[]# R array
C_values=[]# C or alpha array

#append R values and alpha values
for line in R_file:
   line=line.rstrip()
   R_values.append(line)

for line in C_file:
   line=line.rstrip()
   C_values.append(line)


#loops through each R value with the alpha values looped for each
#R values
for i in R_values:
   for j in C_values:
      R_C=open('R_C.dat','w')#opens a new file
      R_C.write(str(i)+'\n'+str(j))#writes R value and alpha values
      R_C.close()# close file
      os.system('./RUN')#runs a program that takes the R and
           #alpha values from the R_C.dat file.
      time.sleep(1)# computer sleeps for one second
```

Single_three_body.cpp

```
#include<iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include "Random_Points.h"//includes tausworthe 88
using namespace std;
//Neon's three body energy function prototype.
double EAT(double &,double &, double &);
double MAT(double &,double &, double &);
double TBE(double &,double &, double &);

int main()
{
  ifstream fp;
  fp.open("R_C.dat");//opens the file that the python script
//created
  unsigned ran_state[4];//setting up the Random point generator
  int i, n=1E6,a=2,b=6;//2 and 6 is the small equilateral
//triangle this could be changed if desired
  a=a-1,b=b-1;//arrays start at zero
 double xapoint,yapoint,zapoint,xbpoint,ybpoint,zbpoint,xcpoint,ycpoint,zcpoint;
 double xa,ya,za,xb,yb,zb,xc,yc,zc,alpha;
 double func,sum=0;
 double R,r1,r2,r3,three_body;
 double xcood[25],ycood[25];
    fp>>R>>alpha;


 RPG Rando;//calling my Random point generator with Rando as my object.

   {

  xcood[0]=0*R;
  xcood[1]=0*R;
  xcood[2]=0*R;
  xcood[3]=0.866025404*R;
  xcood[4]=0.866025404*R;
  xcood[5]=0.866025404*R;
  xcood[6]=0.866025404*R;
  xcood[7]=0.866025404*R;
  xcood[8]=1.732050808*R;
  xcood[9]=1.732050808*R;
  xcood[10]=1.732050808*R;
```

```
  xcood[11]=1.732050808*R;
  xcood[12]=2.598076211*R;
  xcood[13]=2.598076211*R;
  xcood[14]=2.598076211*R;
  xcood[15]=2.598076211*R;
  xcood[16]=2.598076211*R;
  xcood[17]=2.598076211*R;

  ycood[0]=2*R;
  ycood[1]=1*R;
  ycood[2]=-1*R;
  ycood[3]=2.5*R;
  ycood[4]=1.5*R;
  ycood[5]=0.5*R;
  ycood[6]=-0.5*R;
  ycood[7]=-1.5*R;
  ycood[8]=2*R;
  ycood[9]=1*R;
  ycood[10]=0*R;
  ycood[11]=-1*R;
  ycood[12]=2.5*R;
  ycood[13]=1.5*R;
  ycood[14]=0.5*R;
  ycood[15]=-0.5*R;
  ycood[16]=-1.5*R;



    {
          {
           {

           Rando.rsetup(ran_state);
            sum=0;
            three_body=0;
  for (i=0; i<n; i++)

    {
          {
//XYZ for three atoms acquired by getting a Random point
//scaled to a gaussian between 0 and 1
      xapoint = Rando.rangauss(ran_state);
      yapoint = Rando.rangauss(ran_state);
      zapoint = Rando.rangauss(ran_state);
      xbpoint = Rando.rangauss(ran_state);
      ybpoint = Rando.rangauss(ran_state);
```

```
      zbpoint = Rando.rangauss(ran_state);
      xcpoint = Rando.rangauss(ran_state);
      ycpoint = Rando.rangauss(ran_state);
      zcpoint = Rando.rangauss(ran_state);

//The above point are adjusted by alpha the size of the cloud
// and adding the base position of the triangles
      xa=alpha*xapoint+xcood[a];
      ya=alpha*yapoint+ycood[a];
      za=alpha*zapoint+0;
      xb=alpha*xbpoint+0;
      yb=alpha*ybpoint+0;
      zb=alpha*zbpoint+0;
      xc=alpha*xcpoint+xcood[b];
      yc=alpha*ycpoint+ycood[b];
      zc=alpha*zcpoint+0;

//calculating the distance of the sides of the triangles
      r1=sqrt(pow((xb-xa),2)+pow((yb-ya),2)+pow((zb-za),2) );
      r2=sqrt(pow((xc-xb),2)+pow((yc-yb),2)+pow((zc-zb),2) );
      r3=sqrt(pow((xa-xc),2)+pow((ya-yc),2)+pow((za-zc),2) );

//call whatever three body function you would like
      three_body+=MAT(r1,r2,r3);

            }
    }
//prints out average three body energy, alpha and R values
  cout<<three_body/n<<" "<<alpha<<" "<<R<<endl;

    }
              }}}

    fp.close();
  return(0);

            }


//Modified Axilrod Teller function
// I recommend reading the paper
double MAT(double &r1,double &r2, double &r3)
{
//parameters for the function
    double modified_Axilrod,A=566.969,alphatr=1.1896,Ctr=11.835;
    double cosa,cosb,cosc,fo;
```

```
//law of cosines
  cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);

  fo=1+3*cosa*cosb*cosc;

//calculates function
  modified_Axilrod=(-A*exp(-alphatr*(r1+r2+r3))+Ctr/(pow(r1,3)*pow(r2,3)*pow(r3,3)))*fo;


  return modified_Axilrod;//returns three body function

}


//Three Body energy of Ermakova
// I recommend reading the paper
double TBE(double &r1,double &r2, double &r3)
{
//law of cosines
  double cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  double cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  double cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);
  double TBE,p=r1*r2*r3,Q=cosa*cosb*cosc,a1=-1494.04316399,a2=-2.40436522E-2;
  double a3=1.98381726E-4,a4=-3.34763381E4,a5=196.84364784;
  double a6=-1.00003202E-02,a7=4.64397240E-05,a8=-77.45577385;
  double fo=(pow(p,4)*(1.0+a6*p+a7*pow(p,2)));
  double part1,part2,part3;

  part1=a1/(pow(p,4)*(1+a2*p+a3*pow(p,2)));
  part2=Q*(a4+a5*p)/fo;
  part3=pow(Q,2)*a8/pow(p,3);

  TBE=part1+part2+part3;


  return TBE;//returns three body function
}


//Extended Axilrod Teller
// I recommend reading the paper
double EAT(double &r1,double &r2, double &r3)
{
//parameters
```

```
  double fo, rg=pow((r1*r2*r3),(.33333)),rs=r1+r2+r3;
  double c0=12.9236,c1=466.449,c2=-168.680,c3=4.32545,c4=1.23818;
  double cosc,cosb,cosa,three_body;

//law of cosines
  cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);

  fo=1+3*cosa*cosb*cosc;

  three_body=fo*(c0*pow(rg,(-9))+(c1+c2*pow(rg,2)+c3*pow(rg,4))*exp(-c4*rs));

  return three_body;//returns three body energy
}
```

Send_all.py

'''

This program sends calculations for the total three body contributions in a crystal at different R values. It checks the two files for R values and C or alpha values. Then for every R value it will calculate a three body energy for every C or alpha value. Things to watch out for: watch out for blank lines in the R.dat or C.dat, sometimes python will include blank lines into the array. Also make sure you are using the correct alpha value this program takes the gaussian in the form of exp(-(alpha(x^2))).

'''

```
import os
import time
R_file=open('R.dat','r').readlines()#R values
C_file=open('C.dat','r').readlines()#alpha values

R_values=[]# R array
C_values=[]# C or alpha array

#append R values and alpha values
for line in R_file:
   line=line.rstrip()
   R_values.append(line)

for line in C_file:
   line=line.rstrip()
   C_values.append(line)

#loops through each R value with the alpha values looped for each
#R values
for i in R_values:
   for j in C_values:
      R_C=open('R_C.dat','w')#opens a new file
      R_C.write(str(i)+'\n'+str(j))#writes R value and alpha
      R_C.close()#close file
      os.system('./RUN_RPG')#runs a script that takes the R   #and alpha values from the R_C.dat
file.
      time.sleep(1)
```

RUN_RPG.cpp

```
g++ XYZ_coordinates.cpp
./a.out > distances.txt

sleep 1
g++ Triangle_set_up.cpp
./a.out

sleep 1
g++ Remove_duplicates.cpp
./a.out > different_tri.dat

sleep 1

g++ final_cut.cpp
./a.out

sleep 1
g++ Three_body_contribution.cpp
./a.out
```

Triangle_set_up.cpp

```
#include<iostream>
#include<fstream>
#include<math.h>
using namespace std;

int main()
{
  ifstream fp;
  ofstream op;
  fp.open("distances.txt");//opens distances
  op.open("Triangle_configurations.txt");//open a file to ouput
  double distances,X[10000],Y[10000],Z[10000];
  int i=0,number,j=0,count=0;
  double xa,ya,za,xb,yb,zb,xc,yc,zc;
  double R1,R2,R3;

  while(true)
    {
      fp>>distances>>X[i]>>Y[i]>>Z[i];//gets the XYZ's
      if (fp.eof())
        {break;}
      i++;
    }
  number=i;

  xa=0,ya=0,za=0;
  for(i=0;i<number;i++)//Start forming triangles
    {
      for(j=i+1;j<number;j++)//nested loop
           {count++;
//calculate the sides of all the possible triangles
             R1=sqrt(pow((X[i]-X[j]),2)+pow((Y[i]-Y[j]),2)+pow((Z[i]-Z[j]),2));
             R2=sqrt(pow((X[j]),2)+pow((Y[j]),2)+pow((Z[j]),2));
             R3=sqrt(pow((X[i]),2)+pow((Y[i]),2)+pow((Z[i]),2));
            //prints out XYZ for three atoms and the three R values
             op<<xa<<" "<<ya<<" "<<za<<" ";
             op<<X[i]<<" "<<Y[i]<<" "<<Z[i]<<" ";
             op<<X[j]<<" "<<Y[j]<<" "<<Z[j]<<" ";
             op<<R1<<" "<<R2<<" "<<R3<<endl;
           }
    }

  return 0;
}
```

Remove_duplicates.cpp

```
#include<iostream>
#include<fstream>
#include<math.h>
using namespace std;



int main()
{
  ofstream op;
  ifstream fp,fp2;
  fp.open("Triangle_configurations.txt");//opens up file
  double no_duplicates;
  int i=0,number,j=0,N=1E8,k=0,count=0,NUM=1E6;
  double *xa,*ya,*za,*xb,*yb,*zb,*xc,*yc,*zc;
  double *R1,*R2,*R3;
  double *newxa,*newya,*newza,*newxb,*newyb,*newzb,*newxc,*newyc,*newzc;
  double *newR1,*newR2,*newR3;
  int *marker;
//time for some dynamic allocation
  xa=new double[N];ya=new double[N];za=new double[N];xb=new double[N];yb=new
double[N];zb=new double[N];
  xc=new double[N];yc=new double[N];zc=new double[N];R1=new double[N];R2=new
double[N];R3=new double[N];
  marker=new int[N];

  while(true)
    {
//Reads in to the array

fp>>xa[i]>>ya[i]>>za[i]>>xb[i]>>yb[i]>>zb[i]>>xc[i]>>yc[i]>>zc[i]>>R1[i]>>R2[i]>>R3[i];
       marker[i]=0;//all the markers will be zero
      if (fp.eof())
        {break;}
      i++;
    }
  number=i;// counts the lines


  for(i=0;i<number;i++)
    {
      k=1;
      if (marker[i]==1)//this means you have already evaluated this triangle
//therefore continue will make sure it isn't evaluated twice
           {
```

```
            continue;
          }
     for(j=i+1;j<number;j++)//goes to the next atom on the list
          {
//checks if they are the same triangle
            if((R1[i]==R1[j])&&(R2[i]==R2[j])&&(R3[i]==R3[j]))
              {
                marker[j]=1;//sets marker to 1 so we don't revaluate it
                k++;//counts the frequency
              }
          }
//XYZ for all three atoms and the frequency
     cout<<xa[i]<<" "<<ya[i]<<" "<<za[i]<<" "<<xb[i]<<" "<<yb[i]<<" "<<zb[i]<<" "<<xc[i];
     cout<<" "<<yc[i]<<" "<<zc[i]<<" "<<k<<endl;
   }

  delete [] xa;delete [] ya;delete [] za;
  delete [] xb;delete [] yb;delete [] zb;
  delete [] xc;delete [] yc;delete [] zc;
  delete [] R1;delete [] R2;delete [] R3;
   return 0;
}
```

final_cut.cpp

```
#include<iostream>
#include<fstream>
#include<math.h>

using namespace std;
//function prototypes
double eat(double &,double &,double &);
double MAT(double &,double &,double &);
double TBE(double &,double &,double &);

int main()
{ofstream op;
  ifstream fp,fp1,fp2;
  fp2.open("R_C.dat");//gets the R and Alpha values
  fp1.open("different_tri.dat");//gather all the triangles
  op.open("FINAL_CUT.dat");//output file
  double upper,lower,R,r1,r2,r3;
  double *xa,*ya,*za,*xb,*yb,*zb,*xc,*yc,*zc,*Freq,*three_body;
  int N=1E7;
  xa=new double[N];ya=new double[N];za=new double[N];xb=new double[N];yb=new
double[N];zb=new double[N];
  xc=new double[N];yc=new double[N];zc=new double[N];Freq=new double[N];
  three_body=new double[N];
  int i=0,number;

  while(true)
    {//dymamically allocating an arrays
      fp1>>xa[i]>>ya[i]>>za[i]>>xb[i]>>yb[i]>>zb[i]>>xc[i]>>yc[i]>>zc[i]>>Freq[i];
      if (fp1.eof())
        {break;}
      i++;
    }
  number=i;
  i=0;

  for(int j=0;j<number;j++)
    {
      r1=sqrt(pow((xb[j]-xa[j]),2)+pow((yb[j]-ya[j]),2)+pow((zb[j]-za[j]),2) );
      r2=sqrt(pow((xc[j]-xb[j]),2)+pow((yc[j]-yb[j]),2)+pow((zc[j]-zb[j]),2) );
      r3=sqrt(pow((xa[j]-xc[j]),2)+pow((ya[j]-yc[j]),2)+pow((za[j]-zc[j]),2) );
      three_body[j]=TBE(r1,r2,r3);//THREE BODY FUNCTION
//calculates the three body energy for each triangle with
//whatever function you would like.
    }
```

```
  fp2>>R;//reads in R
  r1=R;r2=R;r3=R;//equilateral triangle
  upper=TBE(r1,r2,r3);//THREE BODY FUNCTION

  if (upper<0)
    {lower=upper; upper*=-1.0;}//set a lower and upper bound
  else
    {lower=upper*-1.0;}
  upper/=1000.0;//the 1000 can be adjusted
  lower/=1000.0;//basically if the three body energy is
//at least.1% (absolute value) of the small equilateral triangle we consider that triangle as
significant, however you can adjust that number

  for(int j=0;j<number;j++)
    {
      if ((three_body[j]> upper)||(three_body[j]<lower))
           {op<<xa[j]<<" "<<ya[j]<<" "<<za[j]<<" "<<xb[j]<<" "<<yb[j]<<" "<<zb[j]<<"
"<<xc[j]<<" "<<yc[j]<<" "<<zc[j]<<" "<<Freq[j]<<endl;}//prints out
    }


      //  fp.close();
  fp1.close();
  fp2.close();
  op.close();

  return 0;
}

//Modified Axilrod Teller function
// I recommend reading the paper
double MAT(double &r1,double &r2, double &r3)
{
//parameters for the function
    double modified_Axilrod,A=566.969,alphatr=1.1896,Ctr=11.835;
    double cosa,cosb,cosc,fo;
//law of cosines
  cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);

  fo=1+3*cosa*cosb*cosc;

//calculates function
  modified_Axilrod=(-A*exp(-alphatr*(r1+r2+r3))+Ctr/(pow(r1,3)*pow(r2,3)*pow(r3,3)))*fo;
```

```
  return modified_Axilrod;//returns three body function

}
//Three Body energy of Ermakova
// I recommend reading the paper
double TBE(double &r1,double &r2, double &r3)
{
//law of cosines
  double cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  double cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  double cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);
  double TBE,p=r1*r2*r3,Q=cosa*cosb*cosc,a1=-1494.04316399,a2=-2.40436522E-2;
  double a3=1.98381726E-4,a4=-3.34763381E4,a5=196.84364784;
  double a6=-1.00003202E-02,a7=4.64397240E-05,a8=-77.45577385;
  double fo=(pow(p,4)*(1.0+a6*p+a7*pow(p,2)));
  double part1,part2,part3;
  part1=a1/(pow(p,4)*(1+a2*p+a3*pow(p,2)));
  part2=Q*(a4+a5*p)/fo;
  part3=pow(Q,2)*a8/pow(p,3);

  TBE=part1+part2+part3;
  return TBE;//returns three body function
}
//Extended Axilrod Teller
// I recommend reading the paper
double EAT(double &r1,double &r2, double &r3)
{
//parameters
  double fo, rg=pow((r1*r2*r3),(.33333)),rs=r1+r2+r3;
  double c0=12.9236,c1=466.449,c2=-168.680,c3=4.32545,c4=1.23818;
  double cosc,cosb,cosa,three_body;
//law of cosines
  cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);

  fo=1+3*cosa*cosb*cosc;

  three_body=fo*(c0*pow(rg,(-9))+(c1+c2*pow(rg,2)+c3*pow(rg,4))*exp(-c4*rs));

 return three_body;//returns three body energy
}
```

Three_body_contribution.cpp

```
#include<iostream>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include<time.h>
#include "Random_Points.h"
using namespace std;

/* MAIN PROGRAM ------------------------------------------- */

double eat(double &,double &, double &);
double MAT(double &,double &, double &);
double TBE(double &,double &,double &);

int main()

{
  clock_t t1,t2;
  t1=clock();
  ifstream fp,fp2,fp3;
  fp.open("R_C.dat");
  fp2.open("FINAL_CUT.dat");
   fp3.open("Updated_start_end.txt");
  double start,end,C,R;
  fp>>R>>C;
  fp3>>start>>end;
  unsigned ran_state[4];
  int i, n=1E6,N=1E6,number,j=0;
 double xapoint,yapoint,zapoint,xbpoint,ybpoint,zbpoint,xcpoint,ycpoint,zcpoint;
 double xa,ya,za,xb,yb,zb,xc,yc,zc,alpha=sqrt(1/(2*C));
 double func,sum=0;
 double r1,r2,r3,three_body;
 double xcood[25],ycood[25],three_body_sum=0;
 double *X1,*Y1,*Z1,*X2,*Y2,*Z2,*X3,*Y3,*Z3,*freq;
 RPG Rando;

 X1=new double[N];Y1=new double[N];Z1=new double[N];X2=new double[N];Y2=new
double[N];Z2=new double[N];
 X3=new double[N];Y3=new double[N];Z3=new double[N];freq= new double[N];

 while(true)
   {
     fp2>>X1[j]>>Y1[j]>>Z1[j]>>X2[j]>>Y2[j]>>Z2[j]>>X3[j]>>Y3[j]>>Z3[j]>>freq[j];
     if(fp2.eof())
       {break;}
     j++;
   }
```

```
number=j;

          Rando.rsetup(ran_state);
           sum=0;

for( j=0;j<number;j++)
{
  three_body=0;
 for (i=0; i<n; i++)
  {

    xapoint = Rando.rangauss(ran_state);
    yapoint = Rando.rangauss(ran_state);
    zapoint = Rando.rangauss(ran_state);
    xbpoint = Rando.rangauss(ran_state);
    ybpoint = Rando.rangauss(ran_state);
    zbpoint = Rando.rangauss(ran_state);
    xcpoint = Rando.rangauss(ran_state);
    ycpoint = Rando.rangauss(ran_state);
    zcpoint = Rando.rangauss(ran_state);


    xa=alpha*xapoint+X1[j];
    ya=alpha*yapoint+Y1[j];
    za=alpha*zapoint+Z1[j];
    xb=alpha*xbpoint+X2[j];
    yb=alpha*ybpoint+Y2[j];
    zb=alpha*zbpoint+Z2[j];
    xc=alpha*xcpoint+X3[j];
    yc=alpha*ycpoint+Y3[j];
    zc=alpha*zcpoint+Z3[j];

    r1=sqrt(pow((xb-xa),2)+pow((yb-ya),2)+pow((zb-za),2) );
    r2=sqrt(pow((xc-xb),2)+pow((yc-yb),2)+pow((zc-zb),2) );
    r3=sqrt(pow((xa-xc),2)+pow((ya-yc),2)+pow((za-zc),2) );

    three_body+=TBE(r1,r2,r3);

         }
         three_body_sum+=(three_body/n)*freq[j];

           }

 cout<<three_body_sum<<" "<<R<<" "<<alpha<<endl;
  t2=clock();
  //    cout<<(t2-t1)/CLOCKS_PER_SEC;
```

```
   fp.close();


  return(0);

          }


double eat(double &r1,double &r2, double &r3)
{
  double fo, rg=pow((r1*r2*r3),(.33333)),rs=r1+r2+r3;
  double c0=12.9236,c1=466.449,c2=-168.680,c3=4.32545,c4=1.23818;
  double cosc,cosb,cosa,three_body;

  cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);

  fo=1+3*cosa*cosb*cosc;

  three_body=fo*(c0*pow(rg,(-9))+(c1+c2*pow(rg,2)+c3*pow(rg,4))*exp(-c4*rs));



  return three_body;
}


double MAT(double &r1,double &r2, double &r3)
{

  double modified_Axilrod,A=566.969,alphatr=1.1896,Ctr=11.835;
  double cosa,cosb,cosc,fo;

  cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);

  fo=1+3*cosa*cosb*cosc;

  modified_Axilrod=(-A*exp(-alphatr*(r1+r2+r3))+Ctr/(pow(r1,3)*pow(r2,3)*pow(r3,3)))*fo;


  return modified_Axilrod;

}
```

```
double TBE(double &r1,double &r2, double &r3)
{
  double cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  double cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  double cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);
  double TBE,p=r1*r2*r3,Q=cosa*cosb*cosc,a1=-1494.04316399,a2=-2.40436522E-2;
  double a3=1.98381726E-4,a4=-3.34763381E4,a5=196.84364784;
  double a6=-1.00003202E-02,a7=4.64397240E-05,a8=-77.45577385;
  double fo=(pow(p,4)*(1.0+a6*p+a7*pow(p,2)));
  double part1,part2,part3;

  part1=a1/(pow(p,4)*(1+a2*p+a3*pow(p,2)));
  part2=Q*(a4+a5*p)/fo;
  part3=pow(Q,2)*a8/pow(p,3);

  TBE=part1+part2+part3;


  return TBE;
}
```

AlphaQ.cpp

```
/*
calulate the avg Q^2 for Q1, Q2, Q3; change the R and alpha value accordingly for a small
equilateral triangle
Gaussian is in the format exp(-(x^2)/(2*alpha^2))
*/

#include<iostream>
#include<fstream>
#include<math.h>
#include "Random_Points.h"//random points
using namespace std;

//function proto types
double calc_Q1(double &,double &,double &,double &);
double calc_Q2(double &,double &,double &);
double calc_Q3(double &,double &,double &);

int main()
{
  RPG Rando;
  unsigned ran_state[4];
  Rando.rsetup(ran_state);
  double R=5.92,alpha=.312;
  int whichbin;
  int N=1E6;
  double X1=0,Y1=R,X2=(sqrt(3.0)/2)*R,Y2=.5*R;
  double xa,ya,za,xb,yb,zb,xc,yc,zc;
  double r1,r2,r3,Q1=0,Q2=0,Q3=0;
  double base=sqrt((1.0/3.0))*3*R;//Q1 in base position
  //base=0;

  for (int i=0;i<N;i++)
    {
//XYZ for three atoms plus base positions
            xa = alpha*Rando.rangauss(ran_state)+X1;
            ya = alpha*Rando.rangauss(ran_state)+Y1;
            za = alpha*Rando.rangauss(ran_state)+0;
            xb = alpha*Rando.rangauss(ran_state)+X2;
            yb = alpha*Rando.rangauss(ran_state)+Y2;
            zb = alpha*Rando.rangauss(ran_state)+0;
            xc = alpha*Rando.rangauss(ran_state)+0;
            yc = alpha*Rando.rangauss(ran_state)+0;
            zc = alpha*Rando.rangauss(ran_state)+0;

//sides of triangles
```

```
            r1=sqrt(pow((xb-xa),2)+pow((yb-ya),2)+pow((zb-za),2) );
            r2=sqrt(pow((xc-xb),2)+pow((yc-yb),2)+pow((zc-zb),2) );
            r3=sqrt(pow((xa-xc),2)+pow((ya-yc),2)+pow((za-zc),2) );
//calc Q1^2,Q2^2,Q3^2
            Q1+=pow(calc_Q1(r1,r2,r3,base),2);
            Q2+=pow(calc_Q2(r1,r2,r3),2);
            Q3+=pow(calc_Q3(r1,r2,r3),2);

    }
  //calulate avg Q^2
  cout<<sqrt(Q1)/sqrt(double(N))<<endl;
  cout<<sqrt(Q2)/sqrt(double(N))<<endl;
 cout<<sqrt(Q3)/sqrt(double(N))<<endl;

  return 0;
}


//Q1
double calc_Q1(double &r1,double &r2,double &r3,double &base)
{//Techinically, it is delta T
  double Q1;
  Q1=(1.0/sqrt(3.0))*(r1+r2+r3);//calculate Q1
  Q1=Q1-base;
//subtract Q1 base out, b/c all about that base
  return Q1;
}

//Q2
double calc_Q2(double &r1,double &r2,double &r3)
{
  double Q2;
  Q2=(1.0/sqrt(2.0))*(r2-r3);

  return Q2;
}

//Q3
double calc_Q3(double &r1,double &r2, double &r3)
{
  double Q3;
  Q3=(1.0/sqrt(6.0))*(2*r1-r2-r3);

  return Q3;
}
```

intervals.py

```
'''
This program updates the Q1.dat,Q2.dat and Q3.dat files with the Q values and their R values at
each Q.
'''

from math import sqrt
lowerlimit=7.7#these limits can be adjusted, run the histogram
upperlimit=14.0#program first to determine these limts
R=5.92#R value
Q1interval=[]
Q1=(1.0/sqrt(3.0))*(R*3.0)
intercept=Q1# technically the base

diff=intercept-lowerlimit
diff=diff/25.0# makes 25 intervals from lower limit to the intercept
i=lowerlimit
while i <=(intercept+.001):# the .001 is incase there is any
#fluctuation or precision issues
    Q1interval.append(i)#appends the Q1 values
    i+=diff


diff=upperlimit-intercept
diff=diff/25.0#makes 25 intervals from upper limit to the intercept
i=intercept+diff# I want to start it just above the intercept
while i<=(upperlimit+.001):
    Q1interval.append(i)#appends the rest of the Q1 values
    i+=diff



Q1R_values=[]

for i in Q1interval:# calculates the three R values for each Q1
    RQ1= i*sqrt(3)/3.0
    Q1R_values.append(RQ1)
    Q1R_values.append(RQ1)
    Q1R_values.append(RQ1)


# Q2 limits
lowerlimit=-1.5
upperlimit=1.5
Q2interval=[]
```

```
intercept=0# intercept will be zero for Q2 and Q3

diff=intercept-lowerlimit
diff=diff/25.0# 25 intervals for lower limit to intercept
i=lowerlimit
while i <=(intercept+.001):
    Q2interval.append(i)# append the Q2 values
    i+=diff


diff=upperlimit-intercept
diff=diff/25.0#makes 25 intervals for lower limit to intercept
i=intercept+diff
while i<=(upperlimit+.001):
    Q2interval.append(i)# appends the rest of Q2 values
    i+=diff


Q2R_values=[]

for i in Q2interval:#calculates the R values for each Q2 value
    Q2R1= Q1*sqrt(3.0)/3.0
    Q2R2=(sqrt(2.0)/2.0)*i+Q2R1
    Q2R3=Q2R1-(sqrt(2.0)/2.0)*i
    Q2R_values.append(Q2R1)
    Q2R_values.append(Q2R2)
    Q2R_values.append(Q2R3)



#limits for Q3
lowerlimit=-1.2
upperlimit=1.2
Q3interval=[]

intercept=0  # 0 for Q3

diff=intercept-lowerlimit
diff=diff/25.0#makes 25 intervals from lower limit to intercept
i=lowerlimit
while i <=(intercept+.001):
    Q3interval.append(i)#appends Q3 values
    i+=diff


diff=upperlimit-intercept
```

```
diff=diff/25.0# makes 25 intervals from intercept to upper limit
i=intercept+diff
while i<=(upperlimit+.001):
    Q3interval.append(i)#rest of Q3 values
    i+=diff


Q3R_values=[]

for i in Q3interval:#calc R values for each Q3 value
    Q3R1=sqrt(6.0)*i+sqrt(3.0)*Q1/3.0
    Q3R2=(sqrt(3.0)/2.0)*Q1-Q3R1/2.0
    Q3R3=Q3R2
    Q3R_values.append(Q3R1)
    Q3R_values.append(Q3R2)
    Q3R_values.append(Q3R3)




# output files
Q1_file=open('Q1.dat','w')
Q2_file=open('Q2.dat','w')
Q3_file=open('Q3.dat','w')

j=0
#prints R values and Q values to file
for i in range(0,len(Q1R_values),3):
    Q1_file.write(str(Q1R_values[i])+' '+str(Q1R_values[i+1])+' '+str(Q1R_values[i+2])+'
'+str(Q1interval[j])+'\n')
    Q2_file.write(str(Q2R_values[i])+' '+str(Q2R_values[i+1])+' '+str(Q2R_values[i+2])+'
'+str(Q2interval[j])+'\n')
    Q3_file.write(str(Q3R_values[i])+' '+str(Q3R_values[i+1])+' '+str(Q3R_values[i+2])+'
'+str(Q3interval[j])+'\n')
    j+=1
```

new_interval.py

```
'''
This program takes the Q1 interval from the interval.py program and subtracts out the base to get
a shifted Q1 value
'''
from math import sqrt

fp=open('Q1.dat','r').readlines()#opens Q1
R=5.92#R value
Q1=(sqrt(1.0)/sqrt(3.0))*3*R# calculates base position
#print Q1 with the base Q1 subtracted out
for line in fp:
    parts=line.split()
    newQ1=float(parts[3])-Q1
    print str(newQ1)+',',
```

Qhistograms.cpp

```
/*
prints out the the frequency for 100 slots in Q1,Q2 and Q3
*/

#include<iostream>
#include<fstream>
#include<math.h>
#include "Random_Points.h"
using namespace std;

//function prototypes
double calc_Q1(double &,double &,double &);
double calc_Q2(double &,double &,double &);
double calc_Q3(double &,double &,double &);

int main()
{
 RPG Rando;
 unsigned ran_state[4];
 Rando.rsetup(ran_state);
 double R=5.92,alpha=0.312;
 int whichbin;
 int slotsQ1[100],slotsQ2[100],slotsQ3[100];
  int N=1E6;
  double X1=0,Y1=R,X2=(sqrt(3.0)/2)*R,Y2=.5*R;
  double binwidth=.1;
  double xa,ya,za,xb,yb,zb,xc,yc,zc;
  double r1,r2,r3,Q1,Q2,Q3;

 for(int i=0;i<100;i++)//initializes 100 slots
  {
   slotsQ1[i]=0;
   slotsQ2[i]=0;
   slotsQ3[i]=0;
  }

 for (int i=0;i<N;i++)
  {//XYZ for three atoms
        xa = alpha*Rando.rangauss(ran_state)+X1;
        ya = alpha*Rando.rangauss(ran_state)+Y1;
        za = alpha*Rando.rangauss(ran_state)+0;
        xb = alpha*Rando.rangauss(ran_state)+X2;
        yb = alpha*Rando.rangauss(ran_state)+Y2;
        zb = alpha*Rando.rangauss(ran_state)+0;
        xc = alpha*Rando.rangauss(ran_state)+0;
```

```
            yc = alpha*Rando.rangauss(ran_state)+0;
            zc = alpha*Rando.rangauss(ran_state)+0;
            //distances calculated
            r1=sqrt(pow((xb-xa),2)+pow((yb-ya),2)+pow((zb-za),2) );
            r2=sqrt(pow((xc-xb),2)+pow((yc-yb),2)+pow((zc-zb),2) );
            r3=sqrt(pow((xa-xc),2)+pow((ya-yc),2)+pow((za-zc),2) );

//calculates Q's
            Q1=calc_Q1(r1,r2,r3);
            Q2=calc_Q2(r1,r2,r3);
            Q3=calc_Q3(r1,r2,r3);

//divide the value by the bin width, then the histogram may need to be shifted based on the
distribution
            whichbin=(Q1/binwidth)-30;//the 30 is the shift
            slotsQ1[whichbin]++;//adds one to the bin that it fits in
//repeat
            whichbin=(Q2/binwidth)+25;
          slotsQ2[whichbin]++;
//repeat
            whichbin=(Q3/binwidth)+25;
          slotsQ3[whichbin]++;

    }

prints the frequency of each bin
  for (int j=0;j<100;j++)
    {
      cout<<slotsQ1[j]<<" "<<slotsQ2[j]<<" "<<slotsQ3[j]<<endl;


    }


  return 0;
}


//calculates Q1
double calc_Q1(double &r1,double &r2,double &r3)
{
  double Q1;
  Q1=(1.0/sqrt(3.0))*(r1+r2+r3);

  return Q1;
}
```

```
//calculates Q2
double calc_Q2(double &r1,double &r2,double &r3)
{
  double Q2;
  Q2=(1.0/sqrt(2.0))*(r2-r3);

  return Q2;
}

//calculates Q2
double calc_Q3(double &r1,double &r2, double &r3)
{
  double Q3;
  Q3=(1.0/sqrt(6.0))*(2*r1-r2-r3);

  return Q3;
}
```

Three_body.cpp

```
/*
This program prints out the Q value intervals and their corresponding three body energy. You
first need to run the intervals.py program to update the Q files.
This info will then be used in maple.
*/
#include<iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include<time.h>
#include "Random_Points.h"
using namespace std;

//function prototypes
double eat(double &,double &, double &);
void file_send(ifstream &);

int main()
{
  ifstream fp,fp2,fp3;
  fp.open("Q1.dat");//opens the Q files
  fp2.open("Q2.dat");
  fp3.open("Q3.dat");
//sends files to functions
  file_send(fp);
  file_send(fp2);
  file_send(fp3);

  fp.close();fp2.close();fp3.close();
  return 0;
}



void file_send(ifstream &fp)
{
  double r1,r2,r3,Q[1000],three_body[1000];
  int i=0,number;
  while(true)
    {
      fp>>r1>>r2>>r3>>Q[i];//reads in R values and Q's
     if(fp.eof())
        {break;}
      three_body[i]=eat(r1,r2,r3);//calls EAT to calc 3body
```

```
      i++;

    }
  number=i;
   for(i=0;i<number;i++)
    {
      cout<<Q[i]<<",";//prints Q's
    }
   cout<<endl<<endl;

  for(i=0;i<number;i++)
    {
      cout<<three_body[i]<<",";//prints 3body energies
    }
  cout<<endl<<endl<<endl;
}
//Extended Axilrod teller
double eat(double &r1,double &r2, double &r3)
{//parameters
  double fo, rg=pow((r1*r2*r3),(.33333)),rs=r1+r2+r3;
  double c0=12.9236,c1=466.449,c2=-168.680,c3=4.32545,c4=1.23818;
  double cosc,cosb,cosa,three_body;
//law of Cosines
  cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);

  fo=1+3*cosa*cosb*cosc;

  three_body=fo*(c0*pow(rg,(-9))+(c1+c2*pow(rg,2)+c3*pow(rg,4))*exp(-c4*rs));

  return three_body;//returns three body energies
}
double TBE(double &r1,double &r2, double &r3)
{
  double cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  double cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  double cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);
  double TBE,p=r1*r2*r3,Q=cosa*cosb*cosc,a1=-1494.04316399,a2=-2.40436522E-2;
  double a3=1.98381726E-4,a4=-3.34763381E4,a5=196.84364784;
  double a6=-1.00003202E-02,a7=4.64397240E-05,a8=-77.45577385;
  double fo=(pow(p,4)*(1.0+a6*p+a7*pow(p,2)));
  double part1,part2,part3;

  part1=a1/(pow(p,4)*(1+a2*p+a3*pow(p,2)));
  part2=Q*(a4+a5*p)/fo;
```

```
  part3=pow(Q,2)*a8/pow(p,3);

  TBE=part1+part2+part3;


  return TBE;
}
```

XYZ.cpp

```
/*
prints out the intervals for Cartesian Coordinates and their associated three body energies.
*/
#include<iostream>
#include<fstream>
#include<math.h>
using namespace std;

double EAT(double &,double &,double &);


int main()
{
  double R=5.92;//R value
  int N=51;
  double X1=0,Y1=R,X2=(sqrt(3.0)/2)*R,Y2=.5*R;
  double xa,ya,za,xb,yb,zb,xc,yc,zc;
  double r1,r2,r3;
  double a,b,c,d,e,f,g,h,j;//I need nine intervals
  int count=0;
  double interval[100],Three_body;

    for( double i=-3.5;i<3.51;i+=.14)//intervals
      {
           interval[count]=i;
           cout<<interval[count]<<",";
           count++;
      }


    cout<<endl<<endl<<endl;
    for(int z=0;z<9;z++)//for all 9 coordinates
      {
           a=0.0,b=0.0,c=0.0,d=0.0,e=0.0;
           f=0.0,g=0.0,h=0.0,j=0.0;
  for (int i=0;i<N;i++)//for all 51 points
    {// the code could be a lot more elegant here
//but it works
      if(z==0)
           {a=interval[i];}
     else if(z==1)
           {b=interval[i];}
     else if(z==2)
        {c=interval[i];}
     else if(z==3)
```

```
      {d=interval[i];}
    else if(z==4)
      {e=interval[i];}
    else if(z==5)
      {f=interval[i];}
    else if(z==6)
      {g=interval[i];}
    else if(z==7)
      {h=interval[i];}
    else if(z==8)
      {j=interval[i];}


            xa = X1+a;
            ya = Y1+b;
            za = 0.0+c;
            xb = X2+d;
            yb = Y2+e;
            zb = 0.0+f;
            xc = 0.0+g;
            yc = 0.0+h;
            zc = 0.0+j;

 //calculates sides of the triangles
            r1=sqrt(pow((xb-xa),2)+pow((yb-ya),2)+pow((zb-za),2) );
            r2=sqrt(pow((xc-xb),2)+pow((yc-yb),2)+pow((zc-zb),2) );
            r3=sqrt(pow((xa-xc),2)+pow((ya-yc),2)+pow((za-zc),2) );

//Calculates three body energy
               Three_body=EAT(r1,r2,r3);
               cout<<Three_body<<",";
   }
  cout<<endl<<endl;
   }

  return 0;
}

//Extended Axilrod Teller
double EAT(double &r1,double &r2, double &r3)
{//parameters
  double fo, rg=pow((r1*r2*r3),(.33333)),rs=r1+r2+r3;
  double c0=12.9236,c1=466.449,c2=-168.680,c3=4.32545,c4=1.23818;
  double cosc,cosb,cosa,three_body;
//law of cosines
  cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
```

```
  cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);

  fo=1+3*cosa*cosb*cosc;

  three_body=fo*(c0*pow(rg,(-9))+(c1+c2*pow(rg,2)+c3*pow(rg,4))*exp(-c4*rs));
  return three_body;//returns three body energy
}

double TBE(double &r1,double &r2, double &r3)
{
  double cosc=(pow(r2,2)+pow(r3,2)-pow(r1,2))/(2*r2*r3);
  double cosb=(pow(r2,2)+pow(r1,2)-pow(r3,2))/(2*r2*r1);
  double cosa=(pow(r1,2)+pow(r3,2)-pow(r2,2))/(2*r1*r3);
  double TBE,p=r1*r2*r3,Q=cosa*cosb*cosc,a1=-1494.04316399,a2=-2.40436522E-2;
  double a3=1.98381726E-4,a4=-3.34763381E4,a5=196.84364784;
  double a6=-1.00003202E-02,a7=4.64397240E-05,a8=-77.45577385;
  double fo=(pow(p,4)*(1.0+a6*p+a7*pow(p,2)));
  double part1,part2,part3;

  part1=a1/(pow(p,4)*(1+a2*p+a3*pow(p,2)));
  part2=Q*(a4+a5*p)/fo;
  part3=pow(Q,2)*a8/pow(p,3);

  TBE=part1+part2+part3;


  return TBE;
}
```

ARGON

Send_all.py

```
'''
This program sends calculations for the total two body contributions in a crystal at different R
values.  It checks the two files for R values and C or alpha values.  Then for every R value it will
calculate a three body energy for every C or alpha value.  Things to watch out for: watch out for
blank lines in the R.dat or C.dat, sometimes python will include blank lines into the array.  Also
make sure you are using the correct alpha value this program takes the gaussian in the form of
exp(-(alpha(x^2))).
'''

import os
import time
R_file=open('R.dat','r').readlines()#R values
C_file=open('C.dat','r').readlines()#alpha values

R_values=[]# R array
C_values=[]# C or alpha array

#append R values and alpha values
for line in R_file:
   line=line.rstrip()
   R_values.append(line)

for line in C_file:
   line=line.rstrip()
   C_values.append(line)

#loops through each R value with the alpha values looped for each
#R values
for i in R_values:
   for j in C_values:
      R_C=open('R_C.dat','w')#opens a new file
      R_C.write(str(i)+'\n'+str(j))#writes R value and alpha
      R_C.close()#close file
      os.system('./RUN_RPG')#runs a program that takes the R                    #and alpha values
from the R_C.dat file.
      time.sleep(1)
```

RUN_RPG

```
g++ XYZ_coordinates.cpp
./a.out > distances.txt
sleep 1

python no_duplicates.py

sleep 1

g++ Two_body_contribution.cpp
./a.out
```

Two_body_contribution.cpp

```
#include<iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include "Random_Points.h"
using namespace std;

//Two body potential function prototypes
double GSM(double &);
double Aziz(double &);
double LJ(double &);
double CT(double &);//this is actually the AGY
double S2n(int &,double &,double &);
double factorial(int &);


int main()
{
 ifstream fp,fp1;
 fp.open("Rcoordinates.txt");//opens file
 fp1.open("R_C.dat");
 unsigned ran_state[4];
 RPG Rando;//Random point object
 double R,xbase[200],ybase[200],zbase[200],freq[200],two_body,C,energy,R1;
 double xapoint,yapoint, zapoint,xbpoint,ybpoint,zbpoint,xa,ya,za,xb,yb,zb,Total_energy=0;
 int count=0,n=1E6,end,i;
 fp1>>R1>>C;
 double alpha=sqrt(1/(2*C));//alpha conversion

 while(true)
  {//read in XYZ and frequency of occurence
   fp>>xbase[count]>>ybase[count]>>zbase[count]>>freq[count];
        if (fp.eof())
          {break;}
   count++;

  }
 end=count;

 for(int j=0;j<end;j++)
  {
   two_body=0;
 for (i=0; i<n; i++)
  {//random points scaled to a gaussian from 0 to 1
```

```
      xapoint = Rando.rangauss(ran_state);
      yapoint = Rando.rangauss(ran_state);
      zapoint = Rando.rangauss(ran_state);
      xbpoint = Rando.rangauss(ran_state);
      ybpoint = Rando.rangauss(ran_state);
      zbpoint = Rando.rangauss(ran_state);
//XYZ including alpha and base positions
      xa=alpha*xapoint+xbase[j];
      ya=alpha*yapoint+ybase[j];
      za=alpha*zapoint+zbase[j];
      xb=alpha*xbpoint;
      yb=alpha*ybpoint;
      zb=alpha*zbpoint;
//calc distance or R value
      R=sqrt(pow((xb-xa),2)+pow((yb-ya),2)+pow((zb-za),2) );
      energy=CT(R);//calculate what ever two body function you would like
      two_body+=energy;
    }
  Total_energy+=(two_body/n)*freq[j];//sum up average two body energy

     }

  cout<<Total_energy<<endl;//prints out the total two body contribution

    fp.close();
    fp1.close();
  return(0);

          }

//I recommend reading the papers to all these functions


//AGY
double CT(double &R)
{//parameters
  double A=82.9493,alpha=1.45485,beta=-0.0379929,b=1.62365;
  double G[10][10],C[20];
  double CT,VHF1,VHF2=0,VHF,VC,VC1=0,VC2=0,S2;
  G[1][1]=2.1887E-04;
  G[2][1]=0.13189;
  G[3][1]=3.2949;
  G[1][2]=-1.4886E-02;
  G[2][2]=0.46024;
  G[3][2]=1.7768;
  C[6]=63.752;
```

```
  C[8]=1556.46;
  C[10]=4.944E+04;
  C[12]=2.073E+06;
  C[14]=1.105E+08;
  C[16]=7.248E+09;

//repulsive terms
  VHF1=A*exp((-alpha*R+pow(beta,1)*pow(R,2)));

  for(int i=1;i<3;i++)
    {
      VHF2+=G[1][i]*exp(-G[2][i]*pow((R-G[3][i]),2));
    }

  VHF=VHF1+VHF2;//total repulsion

//attractive terms
  for (int i=1;i<3;i++)
    {
      VC1+=G[1][i]*exp(-G[2][i]*pow((R-G[3][i]),2));
    }

  for (int i=3;i<9;i++)
    {
      S2=S2n(i,R,b);
      VC2+=S2*(C[2*i]/pow(R,(2*i)));
    }
//total Attractiveness
  VC=-(VC1+VC2);
  CT=VHF+VC;//two body


  return CT;//returns two body energy
}

//s2n function called in the CT function
double S2n(int &i,double &R,double &b)
{
  double S2=0,kfact;

  for(int k=0;k<=2*i;k++)
    {
      kfact=factorial(k);//factorial function
      S2+=pow((b*R),k)/kfact;
    }
```

```
  S2=1.0-exp(-b*R)*S2;

  return S2;//returns s2
}


double factorial(int &k)//calculates factorial
{double fact=1.0;
  if (k==0)
    {fact=1;return fact;}
  for (int i=1;i<=k;i++)
    {
      fact*=i;
    }

  return fact;
}

double LJ(double &R)//Lennard Jones
{
//five parameters were found in Literature
//comment out the ones that are not being evaluated
  // double sigma=6.427221172,epsilon=3.78510E-04;//LJ
  //    double sigma=6.42911531,epsilon=3.699156E-04;//LJ 1
  //  double sigma=6.42722117, epsilon=3.800178E-04;//LJ 2
      double sigma=6.35255198, epsilon=4.499886E-04;//LJ 3
  //   double sigma=6.43667297, epsilon=3.793844E-04;//LJ 4
  //    double sigma=6.32325142, epsilon=3.980686E-04;//LJ 5

    double VLJ;

    VLJ=4.0*epsilon*(pow((sigma/R),12)-pow((sigma/R),6));

    return VLJ;//return two body energy

  }

//technically this is the n-6 potential
//However, I didn't like that name so I called it the GSM
// for the first letter of the last names to the people that
//created it
double GSM(double &R)
{//parameters
  double gamma,epsilon=4.482627057E-4;
  double GSM,n,m=13,r,rm=7.100189;
  r=R/rm;
```

```
  n=m+gamma*(r-1);

  GSM=epsilon*(((6.0/(n-6))*pow(r,-n)-(n/(n-6))*pow(r,-6)));

  return GSM;//Returns two body energy

}
//the Aziz-Slaman
double Aziz(double &R)
{//parameters
  double alpha=11.9196,beta=-2.371328,C6=.651991,C8=3.68594,C10=-2.99307;
  double aziz,D=1.36,A=99744.4,rm=7.100189,fx,r,epsilon=4.535639E-4;
  r=R/rm;

  if(r<D)
    {fx=exp(-pow((D/r -1.0),2));}
  else
    {fx=1.0;}


  aziz=A*exp(-alpha*r+beta*pow(r,2))-fx*(C6/pow(r,6)+C8/pow(r,8)+C10/pow(r,10));
  aziz*=epsilon;
  return aziz;//returns two body potential

}
```

send_all.py

```
'''
This program sends calculations for a single small equilateral triangle.  It checks the two files for
R values and C or alpha values.  Then for every R value it will calculate a three body energy for
every C or alpha value.  Things to watch out for: watch out for blank lines in the R.dat or C.dat,
sometimes python will include blank lines into the array.  Also make sure you are using the
correct alpha value this program takes the gaussian in the form of exp(-(x^2) /(2*(alpha^2))).
'''
import os
import time

R_file=open('R_values.dat','r').readlines()#R values
C_file=open('C_values.dat','r').readlines()#alpha values

#append R values and alpha values
for line in R_file:
   line=line.rstrip()
   R_values.append(line)

for line in C_file:
   line=line.rstrip()
   C_values.append(line)

#loops through each R value with the alpha values looped for each
#R values
for i in R_values:
   for j in C_values:
      R_C=open('R_C.dat','w')#opens a new file
      R_C.write(str(i)+'\n'+str(j))#writes R value and alpha values
      R_C.close()# close file
      os.system('./RUN')#runs a program that takes the R and
           #alpha values from the R_C.dat file.
      time.sleep(1)# computer sleeps for one second
```

RUN_RPG

```
g++ cencek.cpp
./a.out > triangles.dat

sleep 1

f95 AR3.f
./a.out < triangles.dat > energy.dat

sleep 1

g++ average.cpp
./a.out
```

Cencek.cpp

```
#include<iostream>
#include<fstream>
#include<math.h>
#include "Random_Points.h"//random points
using namespace std;

int main()
{
  RPG Rando;//random point object
  int N=1E6;
  ifstream fp;
  unsigned ran_state[4];
  double R, C,alpha;
  double xa,ya,za,xb,yb,zb,xc,yc,zc;
  fp.open("R_C.dat");
    fp>>R>>C;//reads in R and C
  Rando.rsetup(ran_state);
  alpha=sqrt(1/(2*C));//converts C to alpha
  double r1,r2,r3;
  double X1=0,Y1=R,X2=(sqrt(3.0)/2)*R,Y2=.5*R;


  cout<<N<<endl;

 for(int i=0;i<N;i++)
    {//XYZ for all three atoms
     xa=alpha* Rando.rangauss(ran_state)+X1;
     ya=alpha* Rando.rangauss(ran_state)+Y1;
     za=alpha* Rando.rangauss(ran_state)+0;
     xb=alpha* Rando.rangauss(ran_state)+X2;
     yb=alpha* Rando.rangauss(ran_state)+Y2;
     zb=alpha* Rando.rangauss(ran_state)+0;
     xc=alpha* Rando.rangauss(ran_state)+0;
     yc=alpha* Rando.rangauss(ran_state)+0;
     zc=alpha* Rando.rangauss(ran_state)+0;

//calculates the sides of the triangles
     r1=sqrt(pow((xb-xa),2)+pow((yb-ya),2)+pow((zb-za),2) );
     r2=sqrt(pow((xc-xb),2)+pow((yc-yb),2)+pow((zc-zb),2) );
     r3=sqrt(pow((xa-xc),2)+pow((ya-yc),2)+pow((za-zc),2) );

     cout<<r1<<" "<<r2<<" "<<r3<<endl;//outputs the sides of the
//triangles
    }
```

```
  fp.close();


  return 0;
}
```

Average.cpp

```
#include<fstream>
#include<iostream>
#include<math.h>

using namespace std;

int main()
{

  double energy,three_body=0;
  ifstream fp;
  int N=0;
  fp.open("energy.dat");//Opens all the three body energy



  while(true)
    {
      fp>>energy;//reads in energy
      if (fp.eof())
           {break;}
      three_body+=energy;//sum up three body energy
      N++;
    }

  cout<<three_body/double(N)<<endl;//average three body energy

  return 0;
}
```

send_all.py

```
'''
This program sends calculations for the total three body contributions in a crystal at different R
values.  It checks the two files for R values and C or alpha values.  Then for every R value it will
calculate a three body energy for every C or alpha value.  Things to watch out for: watch out for
blank lines in the R.dat or C.dat, sometimes python will include blank lines into the array.  Also
make sure you are using the correct alpha value this program takes the gaussian in the form of
exp(-(alpha(x^2))).
'''

import os
import time
R_file=open('R.dat','r').readlines()#R values
C_file=open('C.dat','r').readlines()#alpha values

R_values=[]# R array
C_values=[]# C or alpha array

#append R values and alpha values
for line in R_file:
   line=line.rstrip()
   R_values.append(line)

for line in C_file:
   line=line.rstrip()
   C_values.append(line)

#loops through each R value with the alpha values looped for each
#R values
for i in range(1,2,1):# these numbers can be adjusted depending
# what you are evaluating
   for j in C_values:
      R_C=open('R_C.dat','w')
      R_C.write(str(R_values[i])+'\n'+str(j))#writes R and Alpha value
      R_C.close()
      os.system('./RUN_RPG')#runs a script that takes the R
#and alpha values from the R_C.dat file
      time.sleep(2)
```

RUN_RPG.cpp

```
g++ XYZ_coordinates.cpp
./a.out > distances.txt

sleep 1
g++ Triangle_set_up.cpp
./a.out

sleep 1
g++ Remove_duplicates.cpp
./a.out > different_tri.dat

sleep 1

g++ final_cut.cpp
./a.out

sleep 1

g++ RPG.cpp
./a.out
```

final_cut.cpp

```
/*
If your using this code I'll apoligize in advance.  This will be a little confusing.  Cencek was nice
enough to publish his function, unfortunately he doesn't use a modern programming language.
His code is written in FORTRAN, and since ancient history isn't my best subject there is a lot of
jumping through hoops for this program and the RPG.cpp program to work with the Cencek
function.  I have heard that there is something known as a C++ to FORTRAN wrapper.  That
might be good to look into to improving the efficiency.  If any help is needed to better
understand this program, I can be reached via Skype.
*/

#include<iostream>
#include<fstream>
#include<math.h>
#include<stdlib.h>
using namespace std;

int main()
{
  ofstream op,op2;
  ifstream fp,fp1,fp2,fp3;
  fp.open("energy.dat");
  fp2.open("R_C.dat");
  fp1.open("different_tri.dat");
  op.open("FINAL_CUT.dat");
  op2.open("FROZEN.dat");
  double upper,lower,R,r1,r2,r3;
  double *xa,*ya,*za,*xb,*yb,*zb,*xc,*yc,*zc,*Freq,*three_body;
  int N=1E7,j=0;
  xa=new double[N];ya=new double[N];za=new double[N];xb=new double[N];yb=new
double[N];zb=new double[N];
  xc=new double[N];yc=new double[N];zc=new double[N];Freq=new double[N];
  three_body=new double[N];
  int i=0,number,send=1;
  while(true)
    {//dynamically allocate some arrays
      fp1>>xa[i]>>ya[i]>>za[i]>>xb[i]>>yb[i]>>zb[i]>>xc[i]>>yc[i]>>zc[i]>>Freq[i];
      if (fp1.eof())
        {break;}
      i++;
    }
  number=i;
  i=0;
  op2<<number<<endl;
  for(j=0;j<number;j++)
    {//calculates distances
```

```
      r1=sqrt(pow((xb[j]-xa[j]),2)+pow((yb[j]-ya[j]),2)+pow((zb[j]-za[j]),2) );
      r2=sqrt(pow((xc[j]-xb[j]),2)+pow((yc[j]-yb[j]),2)+pow((zc[j]-zb[j]),2) );
      r3=sqrt(pow((xa[j]-xc[j]),2)+pow((ya[j]-yc[j]),2)+pow((za[j]-zc[j]),2) );
//outputs distances
      op2<<r1<<" "<<r2<<" "<<r3<<endl;
    }


  op2.close();
//run a script that takes the data from FROZEN.dat
//and calculates the three body energy of each one
//then averages it
      system("./FROZEN_Cencek");
      system("sleep 1");
      fp3.open("frozen_data.dat");//open file with the average
//three body energy
      j=0;

  while(true)
            {
      fp>>three_body[j];//reads in each individual Energy from energy.dat
      if (fp.eof())
            {break;}
      j++;
            }

  fp3.close();




  fp2>>R;//reads in R
  r1=R;r2=R;r3=R;

  op2.open("FROZEN.dat");//opens file
  op2<<send<<endl;
  op2<<r1<<" "<<r2<<" "<<r3<<endl;//writes a frozen call
  op2.close();

  system("./FROZEN_Cencek");//sends the the one triangle
//therefore the three body energy of a small equilateral
//triangle to set the bounds
  system("sleep 1");
  fp3.open("frozen_data.dat")
  fp3>>upper;
```

```
  if (upper<0)
    {lower=upper; upper*=-1.0;}
  else
    {lower=upper*-1.0;}
  upper/=1000.0;//the 1000 can be adjusted to any number
  lower/=1000.0;//basically if the three body energy is
//at least.1% (absolute value) of the small equilateral triangle we consider that triangle as
significant, however you can adjust that number.

  for(int j=0;j<number;j++)
    {
      if ((three_body[j]> upper)||(three_body[j]<lower))
           {op<<xa[j]<<" "<<ya[j]<<" "<<za[j]<<" "<<xb[j]<<" "<<yb[j]<<" "<<zb[j]<<"
"<<xc[j]<<" "<<yc[j]<<" "<<zc[j]<<" "<<Freq[j]<<endl;}//print out
    }

  fp.close();
  fp1.close();
  fp2.close();
  op.close();
  fp3.close();

  return 0;
}
```

RPG.cpp

```
//Same deal as the final_cut.cpp program
#include<iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include<time.h>
#include "Random_Points.h"
using namespace std;


int main()

{
  clock_t t1,t2;
  t1=clock();
  ofstream op;
  ifstream fp,fp2,fp3,fp4;
  fp.open("R_C.dat");//R and alpha data
  fp2.open("FINAL_CUT.dat");
   op.open("Frozen.dat");
  double start,end,C,R;
  fp>>R>>C;//reads in R and alpha info
  unsigned ran_state[4];
  int i, n=1E6,N=1E6,number,j=0;
 double xapoint,yapoint,zapoint,xbpoint,ybpoint,zbpoint,xcpoint,ycpoint,zcpoint;
 double xa,ya,za,xb,yb,zb,xc,yc,zc,alpha=sqrt(1/(2*C));//converts C to alpha
 double func,sum=0;
 double r1,r2,r3,three_body;
 double xcood[25],ycood[25],three_body_sum=0;
 double *X1,*Y1,*Z1,*X2,*Y2,*Z2,*X3,*Y3,*Z3,*freq;
 RPG Rando;//Random point object

 X1=new double[N];Y1=new double[N];Z1=new double[N];X2=new double[N];Y2=new
double[N];Z2=new double[N];
 X3=new double[N];Y3=new double[N];Z3=new double[N];freq= new double[N];

 while(true)
   {//dynamically allocating arrays
     fp2>>X1[j]>>Y1[j]>>Z1[j]>>X2[j]>>Y2[j]>>Z2[j]>>X3[j]>>Y3[j]>>Z3[j]>>freq[j];
     if(fp2.eof())
       {break;}
     j++;
   }
 number=j;
```

```
 op<<number*n<<endl;//total number of triangles to be evaluated

            Rando.rsetup(ran_state);
             sum=0;

 for( j=0;j<number;j++)
 {
   three_body=0;
  for (i=0; i<n; i++)
    {
//random points scaled to a gaussian from 0 to 1.
      xapoint = Rando.rangauss(ran_state);
      yapoint = Rando.rangauss(ran_state);
      zapoint = Rando.rangauss(ran_state);
      xbpoint = Rando.rangauss(ran_state);
      ybpoint = Rando.rangauss(ran_state);
      zbpoint = Rando.rangauss(ran_state);
      xcpoint = Rando.rangauss(ran_state);
      ycpoint = Rando.rangauss(ran_state);
      zcpoint = Rando.rangauss(ran_state);

//XYZ for all three atoms
      xa=alpha*xapoint+X1[j];
      ya=alpha*yapoint+Y1[j];
      za=alpha*zapoint+Z1[j];
      xb=alpha*xbpoint+X2[j];
      yb=alpha*ybpoint+Y2[j];
      zb=alpha*zbpoint+Z2[j];
      xc=alpha*xcpoint+X3[j];
      yc=alpha*ycpoint+Y3[j];
      zc=alpha*zcpoint+Z3[j];
//calculate distance
      r1=sqrt(pow((xb-xa),2)+pow((yb-ya),2)+pow((zb-za),2) );
      r2=sqrt(pow((xc-xb),2)+pow((yc-yb),2)+pow((zc-zb),2) );
      r3=sqrt(pow((xa-xc),2)+pow((ya-yc),2)+pow((za-zc),2) );

      op<<r1<<" "<<r2<<" "<<r3<<endl;//outputs triangle configurations
//this is what takes a while
    }
           }

 op.close();

 system("./CENCEK");//calls cencek to convert each triangle to
//a three body energy, which outputs to energy.dat
 system("sleep 1");
```

```
 fp4.open("energy.dat");//opens the energy.dat

//here's where the fun begins
//initialization
 int k=0;
 double energy;
 three_body=0;
 j=0;
 while(true)
    {
      fp4>>energy;//reads in each energy
      if (fp4.eof())
           {break;}

      three_body+=energy;//sum it up
      if((k+1)%n==0)//every 1 million energies
           {
   three_body_sum+=(three_body/n)*freq[j];//add the three body
//every million triangles
   three_body=0;
   j++;
           }
      k++;
    }


   cout<<three_body_sum<<" "<<R<<" "<<alpha<<endl;//prints out
    t2=clock();
    //    cout<<(t2-t1)/CLOCKS_PER_SEC;
    fp.close();
    fp4.close();
  return(0);

}
```

Cencek.cpp

```
#include<iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include<time.h>
#include "Random_Points.h"
using namespace std;

//function prototypes
double eat(double &,double &, double &);
void file_send(ifstream &);

int main()

{
  ifstream fp,fp2,fp3;
//takes the Q data
  fp.open("Q1.dat");
  fp2.open("Q2.dat");
  fp3.open("Q3.dat");
//function calls
  file_send(fp);
  file_send(fp2);
  file_send(fp3);

  fp.close();fp2.close();fp3.close();
  return 0;

          }


void file_send(ifstream &fp)
{
  double r1,r2,r3,Q[1000],three_body[1000];
  double energy;
  int i=0,number;
  ofstream op;
  op.open("Triangles.dat");
  op<<51<<endl;//51 points will be evaluated
  while(true)
    {
      fp>>r1>>r2>>r3>>Q[i];//reads in R values
     if(fp.eof())
        {break;}
```

```
    op<<r1<<" "<<r2<<" "<<r3<<endl;//prints out triangle configurations
    i++;
   }
  op.close();
  system("./CENCEK");// calls cencek to output each 3body energy
//to energy.dat
  system("sleep 1");

  ifstream input;
  input.open("energy.dat");//opens energy.dat
  i=0;
  while(true)
   {
    input>>energy;//reads in 3 body energy
    if (input.eof())
         {break;}
         three_body[i]=energy;//reads 3 body energy into an array
         i++;
   }
  input.close();
  number=i;

   for(i=0;i<number;i++)
   {
    cout<<Q[i]<<",";//prints Q interval
   }
   cout<<endl<<endl;

  for(i=0;i<number;i++)
   {
    cout<<three_body[i]<<",";//prints out corresponding 3 body
   }
  cout<<endl<<endl<<endl;

}
```

VITA

Dan D'Andrea was born in Wynnewood, PA, to the parents of Louis and Karen D'Andrea. He is the last of two siblings. He attended Mother of Divine Providence grade school in King of Prussia, Pennsylvania and Archbishop Carroll High School in Radnor, Pennsylvania. After graduation, he attended Reading Hospital's Nursing school, but later transferred to Kutztown University to pursue a chemistry degree. In summer of 2012, Dan participated in an REU at Xavier University in New Orleans. Under the guidance of Dr. Hua Mei, Dan performed organic synthesis of a monomer for PEM Fuel Cells. He obtained a Bachelor's of Science degree in chemistry from Kutztown University. In Fall of 2013, Dan entered The University of Tennessee, Knoxville, chemistry department. Dan graduated with a Master's of Science degree in Fall 2016 and was hired by The Center for Client Retention.